НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“___” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: «Хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою».

Виконав: студент IV курсу, групи КВ-53
(шифр групи)

Іваненко Антон Романович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник доц.каф.СПСКС, к.т.н., Марченко О. І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з нормоконтролю, доц.каф.СПСКС, к.т.н. Клятченко Я.М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент доц., к.т.н. Заболотня Т.М. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки **6.050102 «Комп'ютерна інженерія»**

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___»_____2019 р.

ЗАВДАННЯ
на дипломний проект студенту
Іваненку Антону Романовичу

1. Тема проекту “Хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою ”.

Керівник проекту: доцент каф. СПіСКС, к.т.н., Марченко О. І.,
затверджені наказом по університету від «22» травня 2019 р. №1330-С

2. Термін подання студентом проекту _____.

3. Вихідні дані до проекту: див. технічне завдання.

4. Зміст пояснювальної записки: огляд існуючих рішень та обґрунтування теми дипломного проекту, технологічні та програмні засоби розробки, хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою, інтерфейс та тестування сервісу.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо):

ІАЛЦ.045450.005 Д1. Архітектура сервісу. Схема структурна
ІАЛЦ.045450.006 Д2. Діаграма схеми БД. Схема структурна

ІАЛЦ.045450.007 Д3. Розподілене блокування. Схема алгоритму
ІАЛЦ.045450.008 Д4. Авторизація користувача. Схема алгоритму

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М.,доцент каф. СПіСКС, к.т.н.		

7. Дата видачі завдання: 15.10.2018.

Календарний план

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання етапів	Примітка
1.	Видача завдання на дипломне проектування	04.11.2018	
2.	Вивчення літератури за тематикою роботи	20.11.2018	
3.	Розробка технічного завдання	10.12.2018	
4.	Аналіз існуючих рішень	20.12.2018	
5.	Розробка структури сервісу	10.01.2019	
6.	Програмна реалізація сервісу	20.03.2019	
7.	Тестування сервісу	25.03.2019	
8.	Підготовка матеріалів текстової частини проекту	10.05.2019	
9.	Підготовка матеріалів графічної частини проекту	20.05.2019	
10.	Попередній огляд дипломного проекту на кафедрі	29.05.2019	

Студент _____ _____
(підпис) (ініціали, прізвище)

Керівник проекту _____ _____
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (56 с., 47 рис. , 4 додатки).

Метою даного дипломного проекту є створення сервісу для гри в шахи з інтерактивним режимом спостереження за грою та подальшою публікацією його у хмарний сервіс.

В роботі розглянуто та проаналізовано існуючі шахові сервіси, їх особливості, переваги та недоліки. Порівняно різні хмарні сервіси і висвітлено актуальність їх використання. Продемонстровано використання технології RPC (Remote Procedure Call) на прикладі розробки онлайн-складової гри в шахи та чату для глядачів. Проектування бази даних для збереження інформації про користувачів було виконано за методологією Code First.

Для реалізації мети дипломного проекту було обрано хмарний сервіс Azure, розроблений компанією Microsoft. Для розробки веб-додатку використано такий стек технологій:

1. Мова програмування C# та фреймворк ASP.NET Core з використанням бази даних MsSQLServer для серверної частини додатку.
2. Мова програмування TypeScript та такі бібліотеки: JQuery, Chessboard.js для реалізації клієнтської частини.

Розроблена система підкріплена інтеграційними тестами, які покривають основні модулі сервісу.

Ключові слова: шахи, хмарний сервіс, ASP.NET Core, C#, RPC, TypeScript, Code Firs, Azure

ABSTRACT

The diploma project includes an explanatory note (56 p., 47 fig., 4 appendices).

The purpose of this graduation project is to create a chess service with an interactive game tracking mode and its subsequent publication in a cloud service.

The work considers and analyzes existing chess services, their features, advantages and disadvantages. Relatively different cloud services and highlighted the relevance of their use. The use of RPC (Remote Procedure Call) technology is demonstrated on the example of developing an online chess and chat component for viewers. Designing a database to store user information was performed according to the Code First methodology.

To realize the goal of the graduation project, the cloud service Azure, developed by Microsoft, was selected. The following technology stack is used to develop a web application:

1. The C # programming language and the ASP.NET Core framework using the MsSQLServer database for the server part of the application.
2. The TypeScript programming language and the following libraries: JQuery, Chessboard.js to implement the client part.

The developed system is backed up by integration tests that cover the main modules of the service.

Keywords: chess, cloud service, ASP.NET Core, C #, RPC, TypeScript, Code Firs, Azure

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.002 ТЗ	Хмарний сервіс гри в	4		
			з інтерактивним			
			спостерігання за грою			
			Технічне завдання			
	А4	ІАЛЦ.045440.003	Хмарний сервіс гри в	1		
			з інтерактивним			
			спостерігання за грою			
			Відомість проекту			
	А4	ІАЛЦ.045440.004 ПЗ	Хмарний сервіс гри в	56		
			з інтерактивним			
			спостерігання за грою			
			Пояснювальна записка			
	А4	ІАЛЦ.045440.005 Д1	Архітектура сервісу	1		
			Схема структурна			
	А4	ІАЛЦ.045440.006 Д2	Діаграма схеми БД	1		
			Схема структурна			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.002 ОА		
Розроб.		Іваненко А.Р.			Хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою	Літ.	Аркуш 1 / Аркушів 2
Перевір.		Марченко О.І					
Н. контр.		Клятченко Я.М.				НТУУ КПІ ім. Ігоря Сікорського ФПМ КВ-53	
Затвер.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.007 Д3	Розподілене блокування	1		
			Схема алгоритму			
	А4	ІАЛЦ.045440.008 Д4	Авторизація користувача	1		
			Схема алгоритму			
		Диск CD-ROM	Матеріали дипломного	1		
			проекту.			

Змін	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.002 ОА	Арк. 2

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ 2

2. ПІДСТАВА ДЛЯ РОЗРОБКИ 2

3. ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ 2

4. ДЖЕРЕЛА РОЗРОБКИ 2

5. ТЕХНІЧНІ ВИМОГИ 2

5.1. Вимоги до програмного продукту, що розробляється 2

5.2. Вимоги до апаратного забезпечення 3

5.3. Вимоги до програмного забезпечення 3

6. ЕТАПИ РОЗРОБКИ 4

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою».

Галузь застосування: досуг, навчання гри у шахи, стеження за новинами.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи ступеня «бакалавр комп'ютерної інженерії», затверджене кафедрою системного програмування і спеціалізованих комп’ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є створення веб-додатку для гри в шахи на мові програмування C# та публікація його у хмарний сервіс.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є технічна та науково-технічна література, технічна документація, публікації у періодичних виданнях та електронні статті у мережі Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

- сумісність з будь-якою операційною системою (Windows, Linux, MacOS);
- можливість спостереження за грою глядачам у випадку, якщо гравці зробили її публічною;
- можливість підписатися на гру з цілю отримувати повідомлення на електронну пошту про результати партії;
- можливість завантажувати зіграні партії на свій пристрій;
- адаптивний дизайн для зручного керування з будь-якого пристрою;

- наявність функціоналу публікації новин для адмістраторів порталу;
- наявність функціоналу блокування гравців за порушення для адміністраторів порталу;
- автоматизована публікація додатку в хмарний сервіс;
- наявність інтеграційних тестів;

5.2. Вимоги до апаратного забезпечення

- Процесор: будь-який 2-ядерний на архітектурі x86/x64 та тактовою частотою 1 Гц.
- Оперативна пам’ять: 512мб та більше;
- Наявність доступу до мережі Internet;

5.3. Ввимоги до програмного забезпечення

- Операційна система Linux Ubuntu 14.x, MacOS 10.12 "Sierra" або Windows 7 SP1
- .NET Core 2.1

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	04.11.2018
2.	Вивчення літератури за тематикою роботи	20.11.2018
3.	Розробка технічного завдання	10.12.2018
4.	Аналіз існуючих рішень	20.12.2018
5.	Розробка структури сервісу	10.01.2019
6.	Програмна реалізація сервісу	20.03.2019
7.	Тестування сервісу	25.03.2019
8.	Підготовка матеріалів текстової частини проекту	10.05.2019
9.	Підготовка матеріалів графічної частини проекту	20.05.2019
10.	Попередній огляд дипломного проекту на кафедрі	29.05.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045440.004 ПЗ	Пояснювальна записка	56		
			Графічна документація			
	А4	ІАЛЦ.045440.005 Д1	Архітектура сервісу	1		
			Схема структурна			
	А4	ІАЛЦ.045440.006 Д2	Діаграма схеми БД	1		
			Схема структурна			
	А4	ІАЛЦ.045440.007 Д3	Розподілене блокування	1		
			Схема алгоритму			
	А4	ІАЛЦ.045440.008 Д4	Авторизація користувача	1		
			Схема алгоритму			
		CD-диск	Матеріали дипломного проекту	1		

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045440.003 ВП		
Розробив		Іваненко А.Р.			*Хмарний сервіс гри в шахи з інтерактивним режимом спостерігання за грою* ***Відомість проекту***	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Марченко О.І.					
Н. контроль		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФПМ, КВ-53	
Затвердив		Тарасенко В.П.					

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ _____ 3
ВСТУП _____ 4
1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ _____ 5
1.1. Аналіз існуючих сервісів для гри в шахи _____ 5
1.2. Обґрунтування теми дипломного проекту та постановка задачі ____ 9
2. ТЕХНОЛОГІЧНІ ТА ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ _____ 10
2.1. Огляд та порівняння сучасних засобів розробки веб-орієнтовних додатків _____ 10
2.1.1 Огляд протоколу HTTP _____ 10
2.1.2 Сучасні засоби розробки клієнтської частини веб-додатку _ 12
2.1.3 Сучасні засоби розробки серверної частини веб-додатку __ 13
2.2. Огляд платформи ASP.NET Core _____ 18
2.2.1 Огляд рушія генерування веб-сторінок Razor _____ 21
2.2.2 Огляд Entity Framework Core _____ 22
2.2.3 Огляд бібліотеки SignalR _____ 24
2.3. Порівняння сучасних хмарних платформ _____ 25
3. ХМАРНИЙ СЕРВІС ГРИ В ШАХИ З ІНТЕРАКТИВНИМ РЕЖИМОМ СПОСТЕРІГАННЯ ЗА ГРОЮ _____ 29
3.1. Загальна архітектура сервісу _____ 29
3.2. Структура бази даних _____ 32
3.3. Розподілений менеджер блокувань _____ 36
3.4. Проектування та розробка сервісу _____ 39

4. ІНТЕРФЕЙС ТА ТЕСТУВАННЯ СЕРВІСУ _____44
4.1. Інтерфейси користувачів _____44
4.1.1 Інтерфейс неавторизованого користувача_____44
4.1.2 Інтерфейс гравця _____47
4.1.3 Інтерфейс адміністратора _____49
4.2. Тестування сервісу _____52
ВИСНОВКИ_____54

ДОДАТКИ

Додаток А. Копії графічних матеріалів

- ІАЛЦ.045450.005 Д1. Архітектура сервісу. Схема структурна
- ІАЛЦ.045450.006 Д2. Діаграма схеми БД. Схема структурна
- ІАЛЦ.045450.007 Д3. Розподілене блокування. Схема алгоритму
- ІАЛЦ.045450.008 Д4. Авторизація користувача. Схема алгоритму

Додаток Б. Фрагмент лістингу програми

- лістинг класу GameDispatcher, який відповідає за комунікацію клієнта за сервером під час гри в шахи
- лістинг класу GameHub, який відповідає за обробку запитів під час гри в шахи

Додаток В. Презентація

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

SSR – Server Side Rendering

SSS – Server Side Scripting

CSR – Client Side Rendering

HTTP – HyperText Transfer Protocol

URI – Uniform Resource Identifier

ASP – Active Server Pages

MVC – Model View Controller

DI – Dependency Injection

БД – База даних

СУБД – Система Управління Базами Даних

CRUD – Create, read, update, delete

RPC – Remote Procedure Call

CQS – Command Query Separation

ВСТУП

Зі збільшенням кількості користувачів з кожним днем, завантаженість WEB-додатків швидко росте. Саме через це, все більше й більше компаній починають використовувати хмарні технології. Тому розробка хмарних сервісів націлених на велику кількість споживачів є дуже актуальною.

Хмарні обчислення, що розповсюджуються провайдерами за моделлю інфраструктура як послуга, дозволяють розробникам не піклуватися про навантаження серверів і нарощування швидкодії апаратного забезпечення, а сконцентруватися на розробці програмного продукту.

Популярність шахів у наш час така велика, що ніхто не може сказати точну цифру кількості тих, хто полюбляє цю гру. За неточною оцінкою ФІДЕ (Міжнародної шахової організації) на сьогодні є близько 300 мільйонів професійних гравців, а це викликає високу популярність шахових сервісів для онлайн гри в шахи.

Отже, таким чином, дослідження та розробки, що пов’язані з проектуванням і розробкою хмарного сервісу для гри в шахи, який би легко масштабувався в залежності від кількості його користувачів та мав високі інтерактивні можливості, що дозволять гравцям взаємодіяти один з одним, є актуальними.

1. ОГЛЯД ІСНУЮЧИХ РІШЕНЬ ТА ОБГРУНТУВАННЯ ТЕМИ ДИПЛОМНОГО ПРОЕКТУ

1.1 Аналіз існуючих сервісів для гри в шахи

Сьогодні існує велика кількість онлайн сервісів для гри в шахи. Кожен з них має як переваги, так і недоліки. Тому для аналізу було обрано три найпопулярніші ресурси світу, а саме – Chess.com, lichess та chess24. Проаналізуємо кожний з них за такими критеріями:

- зручність керування;
- можливість грати партії без глядачів чи з ним;
- наявність внутрішнього чату для гравців та для глядачів;
- аналіз партій ;
- можливість відстежувати гру, якщо користувач не на порталі;
- можливість завантажити гру;
- ціна користування сервісом.

Chess.com був розроблений у 2007 році і продовжує швидко розвиватися. Він має додатки для мобільних пристроїв на базі операційних систем Android, iOS та Windows Phone. До переваг цього ресурсу можна віднести:

- зручність керування – інтерфейс інтуїтивно зрозумілий (рис. 1.1)

Рисунок 1.1 – Інтерфейс гри в шахи на порталі Chess.com

- кожну зіграну партію можна проаналізувати за допомогою рушія;
- наявність чату для гравців;
- можливість завантажити зіграну партію на свій носій;
- наявна українська локалізація;
- ціна платних акаунтів починається від 3 доларів на місяць та є сприйнятою для звичайного мешканця України (рис. 1.2).

Рисунок 1.2 – Ціни на підписки сервісу Chess.com

Але можна зазначити такі недоліки:

- деякі функціональні можливості, як аналіз партій, обмежений і повністю реалізований для платних акаунтів;
- глядач не може відстежувати цікаву йому гру, якщо за обставинами не має можливості зайти на сервіс.

Сервіс lichess був написаний як відкрите програмне забезпечення і є другим за популярністю в світ. Має такі переваги:

- безкоштовний та не має реклами;
- можливий аналіз зіграних партій;
- зручний у керуванні (рис. 1.3);
- глядачі можуть спостерігати за грою та переписуватися в чаті

Рисунок 1.3 – Інтерфейс гри в шахи на порталі lichess.org

- наявна можливість завантажити партію;
- має українську локалізацію, але деякі частини не перекладені.

До недоліків можна віднести:

- глядач не може відстежувати цікаву йому гру, якщо за обставинами не має можливості зайти на сервіс;
- усі партії публічні, не можливо приховати гру.

Chess24 є відносно новим сервісом, який був запущений у 2014 році. Зараз офіційно представлений у таких країнах, як Франція, Англія, Німеччина та Іспанія. Має мобільні додатки для таких операційних систем: IOS та Android. Цей сервіс має такі переваги:

- вбудований рушій для аналізу партій;
- наявна можливість завантажувати партії ;
- глядачі можуть спостерігати за грою;
- має чат для глядачів;
- наявний зручний інтерфейс (рис. 1.4);
- має часткову українську локалізацію.

Але має такі недоліки:

- деякі функціональні властивості, як аналіз партій та завантажування партій, обмежений і повністю реалізований для платних акаунтів;

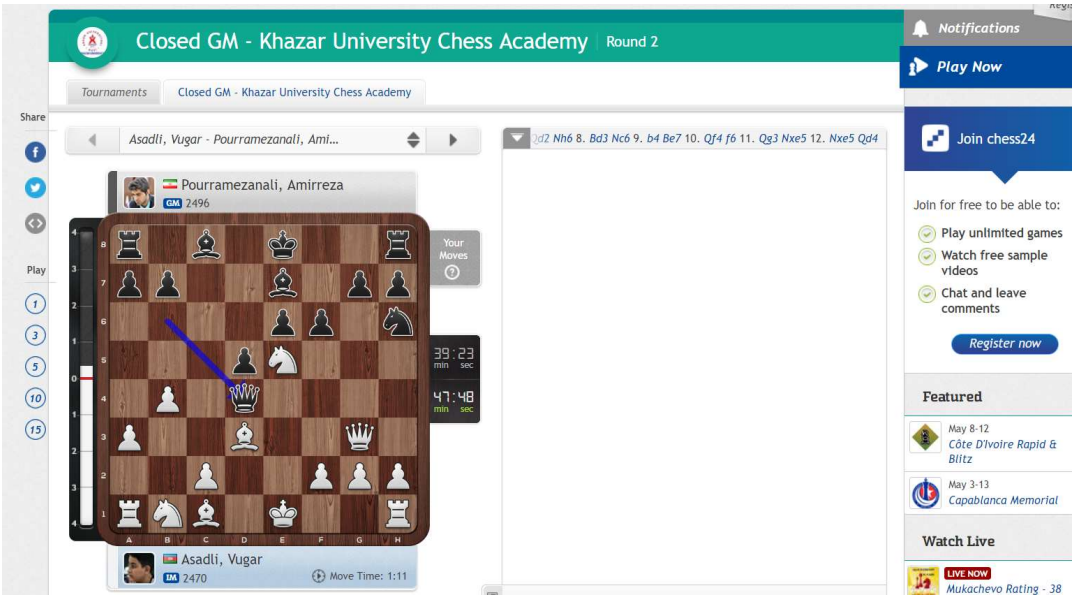

Рисунок 1.4 – Інтерфейс гри в шахи на порталі chess24

- ціна преміум акаунтів близько 10 доларів на місяць є достатньо високою для звичайного мешканця України (рис. 1.5);
- відсутня українська локалізація;
- глядач не може відстежувати цікаву йому гру, якщо не знаходиться на порталі.

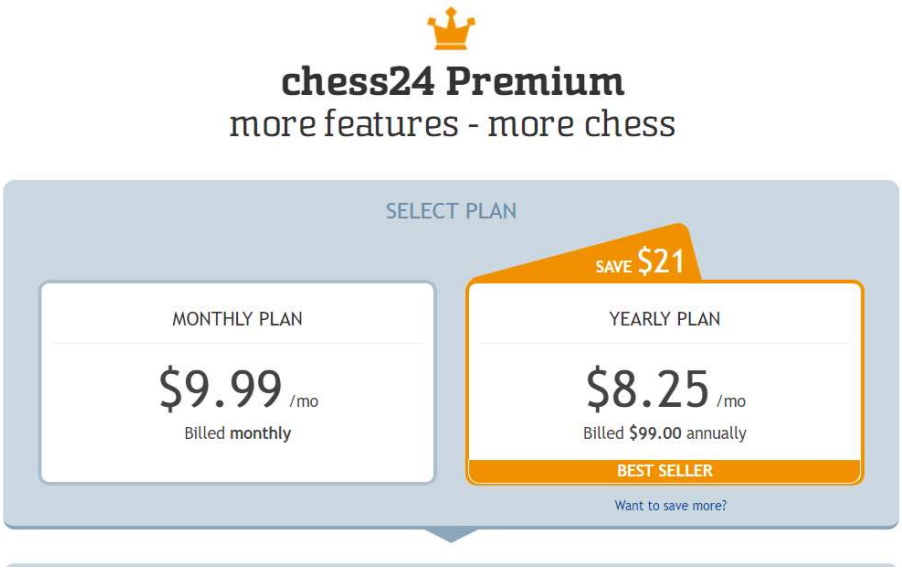

Рисунок 1.2 – Ціни на підписку сервісу Chess24

Варто зазначити, що українських аналогів даних сервісів знайти не вдалося, тому під час порівняння даної трійки лідерів враховувалася наявність української мови та доступність платних можливостей для мешканців України.

1.2 Обґрунтування теми дипломного проекту та постановка задачі

Перш ніж приступати до реалізації програмного продукту, необхідно визначити функції, які повинна виконувати програма.

Виходячи з проведеного аналізу існуючих сервісів для гри в шахи можна підкреслити, що з метою популяризації шахів в нашій країні та враховуючи відсутність україінізованих систем з повністю реалізованими зазначеними вище функціональними властивостями, розробка хмарного сервісу гри в шахи для українського ринку є доцільною.

Сформулюємо функціональні властивості сервісу, які орієнтовані на користувача:

- створення як публічних, так і приватних партій;
- можливість підписатись на цікаву гру зі сповіщенням на електронну пошту;
- завантаження зіграних партій;
- можливість переписуватись у чаті під час гри як з глядачами та в окремому чаті зі своїм суперником.

Функціональні властивості адміністратора сервісу є наступними:

- керування розділом статей;
- блокування користувачів за порушення правил.

Планується, що даний сервіс буде розміщено на хмарній платформі, що забезпечить зручне масштабування під час великого навантаження. Відсутність мобільного додатку можна компенсувати адаптивним дизайном.

2. ТЕХНОЛОГІЧНІ ТА ПРОГРАМНІ ЗАСОБИ РОЗРОБКИ

2.1. Огляд та порівняння сучасних засобів розробки веб-орієнтовних додатків

У наш час кожен з розробників використовує власний інструментарій для розробки додатків, який допомагає виконати роботу швидко та якісно. Тому існує велика кількість мов програмування, бібліотек та платформ націлених на розробку клієнт-серверних програмних продуктів. Такі засоби розробки можна поділити на ті, що відповідають за «front end» та ті, що відповідають за

«back end».

«Front end» – це клієнтська сторона, призначений для користувача інтерфейс до програмно-апаратної частини сервісу.

«Back end» – це програмно-апаратна частина сервісу.

Терміни з'явилися в програмної інженерії внаслідок розвитку принципу поділу відповідальності між зовнішнім поданням і внутрішньої реалізацією.
Взаємодія клієнтської та серверної частин між собою у більшості випадків виконується за допомогою протоколу HTTP.

2.1.1. Огляд протоколу HTTP

HTTP (HyperText Transfer Protocol) – протокол прикладного рівня передачі даних спочатку – у вигляді гіпертекстових документів в форматі «HTML», зараз використовується для передачі довільних даних. Обмін повідомленнями йде по звичайній схемі «запит-відповідь» [2]. Для ідентифікації ресурсів HTTP використовує глобальні Uniform Resource Identifier (URI). На відміну від багатьох інших протоколів, HTTP не зберігає свого стану. Це означає відсутність збереження проміжного стану між парами «запит-відповідь».
URI — компактний рядок літер та символів, який однозначно ідентифікує окремий абстрактний чи фізичний ресурс. Основне призначення таких ідентифікаторів — зробити можливою взаємодію з представленнями ресурсів через мережу.

Протокол HTTP має такі основні методи, які вказують на певну операцію над ресурсом:

- OPTIONS – використовується для визначення можливостей веб-сервера або параметрів з'єднання для конкретного ресурсу;
- GET – використовується для запиту вмісту зазначеного ресурсу;
- HEAD – аналогічний до методу GET, але відрізняється тим, що у відповіді з серверу відсутнє «тіло».
- POST – використовується для передачі даних на сервер;
- PUT – завантажує дані на сервер;
- DELETE – видаляє дані з серверу.
- TRACE – повертає запит так, що клієнт може побачити, яку інформацію проміжні сервери додають або змінюють у запиті.

Протокол HTTP складається з трьох основних частин:

1) стартовий рядок – вказує на тип повідомлення;
2) заголовки – характеризують тіло повідомлення, параметри передачі та інші опції;
3) тіло – містить дані повідомлення.

У відповідь на HTTP-запит сервер повертає відповідь, що містить у стартовому рядку код стану. Код стану є цілим числом, яке складається з 3 десяткових цифр. Перша цифра вказує на клас стану.

Приклад HTTP-запиту та HTTP-відповіді зображено на рис 2.1.

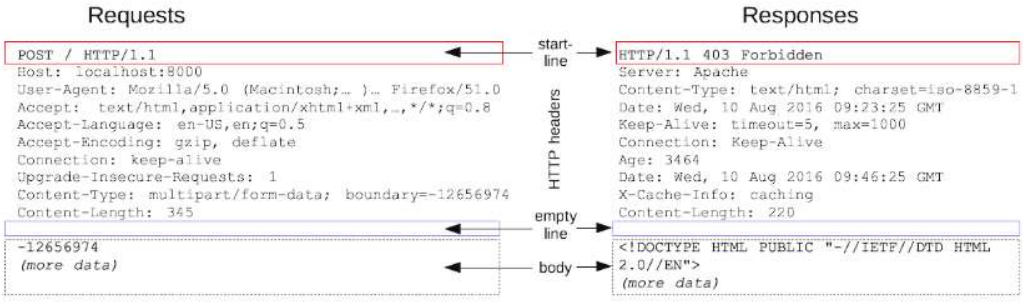

Рисунок 2.1 – HTTP-запит методу POST та HTTP-відповідь з кодом відмови в доступі

2.1.1 Сучасні засоби розробки клієнтської частини веб-додатку

До засобів розробки, націлених на написання клієнтської частини можна віднести такі мови розмітки та програмування:

- HTML (HyperText Markup Language) – мова розмітки гіпертекстових документів. Є стандартом для створення веб-сторінок та веб-орієнтовних додатків;
- CSS (Cascading Style Sheets) – каскадні таблиці стлів, спеціальна мова, розроблена для опису зовнішнього огляду веб-сторінок;
- JavaScript – це об'єктно-орієнтована мова програмування з динамічною типізацією [1]. Реалізація стандарту ECMAScript. Спочатку була розроблена для виконання сценаріїв веб-сторінок. Пізніше набула великою популярності та стала однією з провідних мов програмування.

Разом ці мови утворюють так звану тріаду основних технологій для всесвітньої інтернет мережі. На цьому стеці зараз створено велику кількість бібліотек, які полегшують розробку інтерфейсу веб-додатку.

Розрізняються два фундаментальні підходи до розробки клієнтської частини: рендеринг сторінок на стороні клієнту та рендеринг сторінок на стороні серверу.

Server Side Rendering (SSR) або Server Side Scripting (SSS) – це створення готової для використання та заповненої певними даними веб-сторінки на стороні серверу. Для реалізацію цього сценарію існують серверні скриптові мови та технології, які буде розглянуто в наступному підрозділі 2.1.2.

Client Side Rendering (CSR) – це надання вмісту веб-сторінки на стороні клієнту (браузера у даному випадку) за допомогою мови JavaScript. Таким чином сервер повертає HTML сторінку з базовим вмістом та посиланням на файл скрипта, який відповідно заповнить її інформацією.

Основною проблемою використання мови JavaScript є те, що вона має слабку динамічну типізацію, що значно ускладнює розробку великих додатків.

Динамічна типізація – підхід, який широко використовується в мовах програмування, при якому змінна зв'язується з типом в момент присвоєння значення, а не в момент оголошення змінної.

Слабка типізація – вид типізації в мовах програмування, який дозволяє виконувати безліч неявних перетворень типів автоматично, навіть якщо може відбутися втрата точності або перетворення неоднозначне.

Для усунення цих недоліків у 2012 році компанія Microsoft представила мову програмування TypeScript, яка інтерпретується у мову JavaScript. Дана мова була розроблена Андерсом Хейлсбергом, автором таких мов програмування, як Turbo Pascal, Delphi та C#. TypeScript є розширенням стандарту ECMAScript 5, так як були додані такі функціональні можливості:

- анотації типів та перевірка їх на етапі компіляції;
- вивід типів;
- класи;
- інтерфейси;
- перелічувальні типи;
- модулі;
- кортежі,
- додаткові параметри та параметри за замовчуванням.

Завдяки цим нововведенням у порівнянні з «чистим» JavaScript, дана мова програмування дозволяє легко писати тим, хто звик до сильної статичної типізації, наприклад, для розробників, які працюють с Java або C#, а також забезпечує знаходження помилок в коді ще на етапі компіляції, що можна побачити на рис 2.2 та 2.3.

2.1.1 Сучасні засоби розробки серверної частини веб-додатку

Засобів для розробки серверної частини додатку на сьогоднішній день створено велику кількість. Майже кожна мова програмування має свою бібліотеку чи фреймворк для створення веб-застосунків. Для Python такою

технологією є Django, для мови Java – Java Server Pages (JSP), для C# – фреймворк ASP.NET.

```
// Basic JavaScript
function getPassword(clearTextPassword) {
    if(clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // "password"
```

Рисунок 2.2 – Приклад автоматичного приведення типів в мові програмування JavaScript

```
// Written with TypeScript
function getPassword(clearTextPassword: boolean) : string {
    if(clearTextPassword) {
        return 'password';
    }
    return '********';
}

let password = getPassword('false'); // throws: error TS2345: Argument of type '"false"' i
s not assignable to parameter of type 'boolean'.
```

Рисунок 2.3 – Приклад сповіщення про помилку компілятора TypeScript

Деякі мови програмування були спеціально створені для розробки серверних веб-додатків. Наприклад, PHP, яка вже давно себе зарекомендувала в цій галузі.

Таким чином для порівняння було відібрано такі три технології: мова програмування PHP, Python у поєднанні з фреймворком Jango та мова C# у поєднанні з крос-платформним фреймворком ASP.NET Core, який зараз стрімко стає популярним.

PHP був створений в 1994 році датським програмістом Расмусом Лердорфом. Є скриптовою мовою [3] та виконується інтерпретатором на мові C. І з самого виникнення PHP (скорочення від PHP: Hypertext Preprocessor –

PHP: препроцесор гіпертексту) представляв зручний набір інструментів для спрощеного створення веб-сайтів і веб-додатків. Має слабку динамічну типізацію.

До переваг цієї мови можна віднести:

- багатоплатформність, для всіх найбільш поширених операційних систем (Windows, MacOS, Linux) є свої версії пакетів розробки;
- може працювати з різними веб-серверами: Apache, Nginx, IIS;
- простий та легкий для вивчення;
- підтримує роботу з великою кількістю систем баз даних (MySQL, MSSQL, Oracle, Postgre, MongoDB та інші);
- відкритість вихідного коду.

Але має такі недоліки:

- незручність дизайну мови;
- відсутня зворотна сумісність з попередніми версіями мови;
- повільність мови, яку потрібно компенсувати кешуванням.

Мова Python – це інтерпретована об'єктно-орієнтована мова програмування з сильною динамічною типізацією. Гвідо ван Россумом розробив її в 1990 році. Структури даних високого рівня разом із динамічною семантикою та динамічним зв'язуванням роблять її привабливою для швидкої розробки програм, а також як засіб поєднування наявних компонентів. Python має модульну архітектуру, що сприяє повторному використанню коду. Інтерпретатор мови Python та стандартні бібліотеки доступні розробнику як у скомпільованому вигляді, так і у вихідній формі на всіх основних операційних системах.

Django – це веб-платформа з відкритим вихідним кодом на основі Python, яка слідує архітектурному шаблону Model-View-Template (MVT). Ця платформа була створена у 2003 року, коли веб-програмісти газети Lawrence Journal-World, Адріан Головати та Саймон Уіллісон, почали використовувати Python для розробки додатків. Даний фреймворк був публічно випущений під

ліцензією BSD у липні 2005 року.

Django має такі переваги:

- захищеність, має функціональні можливості запобігання таким видам атак, як SQL ін’єкції чи міжсайтовий скриптинг;
- гнучкий, дану платформу можна використовувати за різними сценаріями;
- простота мови Python;
- вбудовані можливості абстракції доступу до баз даних (БД).

Але можна виділити такі недоліки:

- регулярні вирази для визначення URL, що робить додаток значно важчим;
- розвивається повільно через зворотну сумісність;
- монолітний, закладена певна архітектура і відсутній простір для її корегування.

Мова програмування C# є об'єктно-орієнтованою зі сильною статичною системою типізації для платформи .NET. Розроблена Андерсом Гейлсбергом, Скотом Вілтамутом та Пітером Гольде під егідою Microsoft Research у 2001 році.

Платформа ASP.NET Core представляє технологію від компанії Microsoft, призначену для створення різного роду веб-додатків: від невеликих веб-сайтів до великих веб-порталів і веб-сервісів.

З одного боку, ASP.NET Core є продовженням розвитку платформи ASP.NET. Але з іншого боку, це не просто черговий реліз. Вихід ASP.NET Core фактично означає революцію всієї платформи, її якісна зміна.

Дана платформа має такі переваги:

- наявна конфігурація для спрощеного використання в хмарі;
- можливість розробки і розгортання додатків ASP.NET на Windows, Mac і Linux;
- вбудована підтримка впровадження залежностей (Dependency injection);
- open-source та активний розвиток.

До недоліків можна віднести:

- не всі функціональні можливості перенесені з оригінального ASP.NET;
- складність мови C# у порівнянні с Python та PHP.

Для порівняння швидкодії описаних платформ звернемось до бенчмарку веб-фреймворків від компанії TechEmpower. Проаналізуємо такі основні характеристики:

- швидкість сериалізації/десериалізації в/з формату JSON;
- швидкість передачі тексту.

Результати швидкодії роботи з JSON продемонстровано на рис. 2.4.

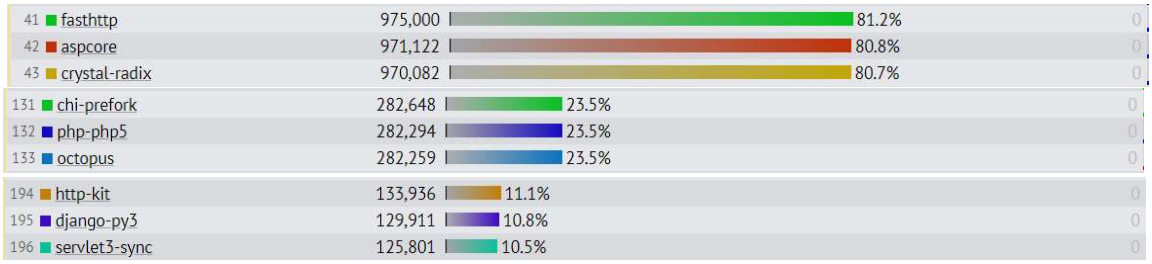

Рисунок 2.4 – Порівняння швидкості роботи з форматом JSON (більше краще)

За 100% було обрано бібліотеку Colossus для мови Scala, як найкраще існуюче рішення. З результату видно, що ASP.NET Core (під номером 43) має кращі вбудовані можливості для роботи з заданим форматом ніж PHP (номер 132) та Django (номер 195).

Результати швидкості передачі тексту продемонстровано на рис 2.5.

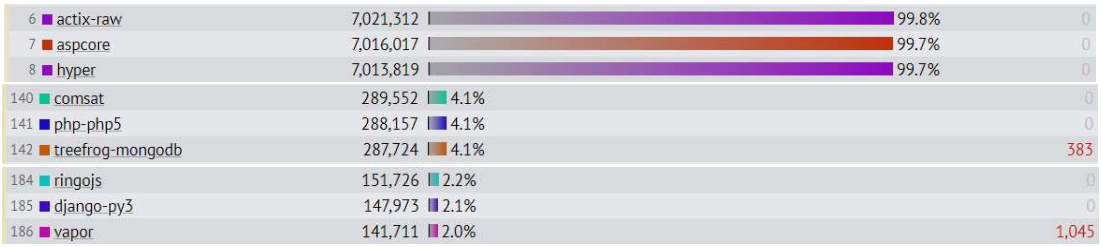

Рисунок 2.5 – Порівняння швидкості передачі тексту (більше краще)

За 100% було обрано платформу ulib для мови C++, як найкраще існуюче рішення. З результату порівняння видно, що ASP.NET Core (під номером 7) майже наблизився до кращого результату, у той час як PHP (номер 141) та Django (номер 185) значно програють по швидкодії.

Таким чином, проаналізувавши подану вище інформацію було вирішено обрати платформу ASP.NET Core для написання заданого сервісу, як найкращу за швидкодією та найбільш орієнтовну для розробки та публікації розробленого додатку на хмарний сервіс. Для написання скриптів на клієнті було обрано мову програмування TypeScript, так як вона за рахунок статичної типізації дозволяє запобігти виникненню потенційних помилок.

2.2. Огляд платформи ASP.NET Core

Розробка над платформою почалася ще в 2014 році. Тоді платформа умовно називалася ASP.NET vNext. У червні 2016 року вийшов перший реліз платформи. На сьогоднішній день останньою стабільною версією є 2.2.

Даний фреймворк працює поверх крос-платформного середовища .NET Core, яке може бути запущено на основних популярних операційних системах: Windows, Mac OS X, Linux. Таким чином, ASP.NET Core дозволяє створювати крос-платформні додатки. І хоча Windows як середовище для розробки і розгортання програми досі позиціонує себе, як основне, але тепер розробник не обмежений тільки цією операційною системою. Тобто він може запускати веб-додатки не тільки на ОС Windows, але і на Linux і Mac OS. А для розгортання веб-застосунків можна використовувати традиційний IIS сервер, або крос-платформний веб-сервер Kestrel.

Завдяки модульності фреймворка всі необхідні компоненти веб-додатки можуть завантажуватися як окремі модулі через пакетний менеджер Nuget.

ASP.NET Core включає в себе фреймворк MVC, який об'єднує функціональність MVC, Web API і Web Pages. У попередніх версіях платформи дані технології реалізувалися окремо і тому містили багато дублюючої функціональності. Зараз же вони об'єднані в одну програмну модель ASP.NET Core MVC.

Дана програмна модель реалізує патерн Model-View-Controller (MVC). Цей шаблон проектування дозволяє поділити дані додатку, користувацький інтерфейс та керуючу логіку на три окремих компоненти: модель, вигляд та

контролер (рис 2.6). Завдяки цьому модифікація кожного компоненту може здійснюватися незалежно один від одного.

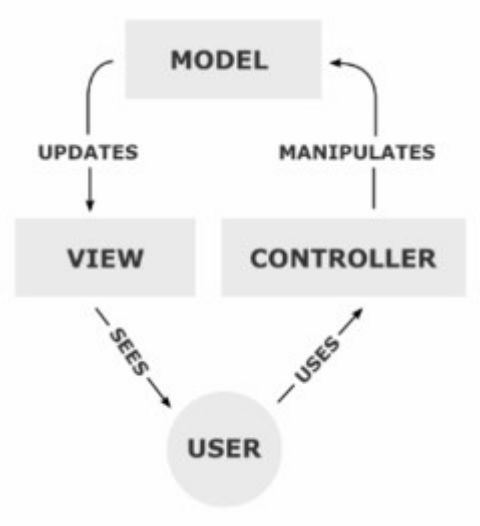

Рисунок 2.6 – Схема взаємодії компонентів MVC

Розподіл обов'язків серед компонентів патерну:

- Модель (Model) представляє дані та реагує на команди контролеру змінюючи свій стан.
- Вигляд (View) виконує відображення даних моделі користувачу, реагуючи на зміни моделі.
- Контролер (Controller) виконує дії користувача та сповіщає модель о наявності змін.

Розглянемо структуру проекту, що розробляється на платформі ASP.NET Core.

Головним класом в даному фреймворку є Startup. Він містить в собі:

- вже налаштовані сервіси, які потрібні для роботи програми;
- конвеєр обробки запитів;
- код для налаштування (або реєстрації) додаткових сервісів. Наприклад, об'єкт контексту Entity Framework Core (див. підрозділ 2.2.2) є сервісом;
- код для налаштування конвеєра обробки запитів.

ASP.NET Core містить в собі вбудовані функціональні можливості для

впровадження залежностей (dependency injection, далі DI). Це шаблон проектування програмного забезпечення, що використовує «інверсію управління» для розв'язання (отримання) залежностей під час надання зовнішньої залежності програмному компоненту.

Впровадження — це передача залежності (сервісу) залежному об'єкту (клієнту).

Існує три способи виконання DI:

1) впровадження в конструктор;
2) впровадження у властивість;
3) впровадження в метод.

ASP.NET Core підтримує перший та третій способи, так як другий у більшості випадків вважається антипатерном.

Для обробки HTTP-запитів та HTTP-відповідей платформа ASP.NET Core має спеціальний конвеєр, який називається Middleware (рис 2.7). Кожний компонент якого виконує наступні функції:

- вирішує, чи передавати запит наступному компоненту в конвеєрі;
- виконує роботу до і після наступного компоненту.

У класі Startup додані наступні компоненти Middleware в певному порядку, який забезпечує уникнення конфліктів:

1) обробник помилок;
2) обробник HTTP перенаправлення;
3) обробник статичних файлів серверу;
4) обробник політики щодо файлів cookie;
5) обробник автентифікації користувача;
6) обробник сесії користувача;
7) обробник MVC.

Все, що було описано вище налаштовує та запускає хост. Хост відповідає за запуск програми та керує її циклом життя.

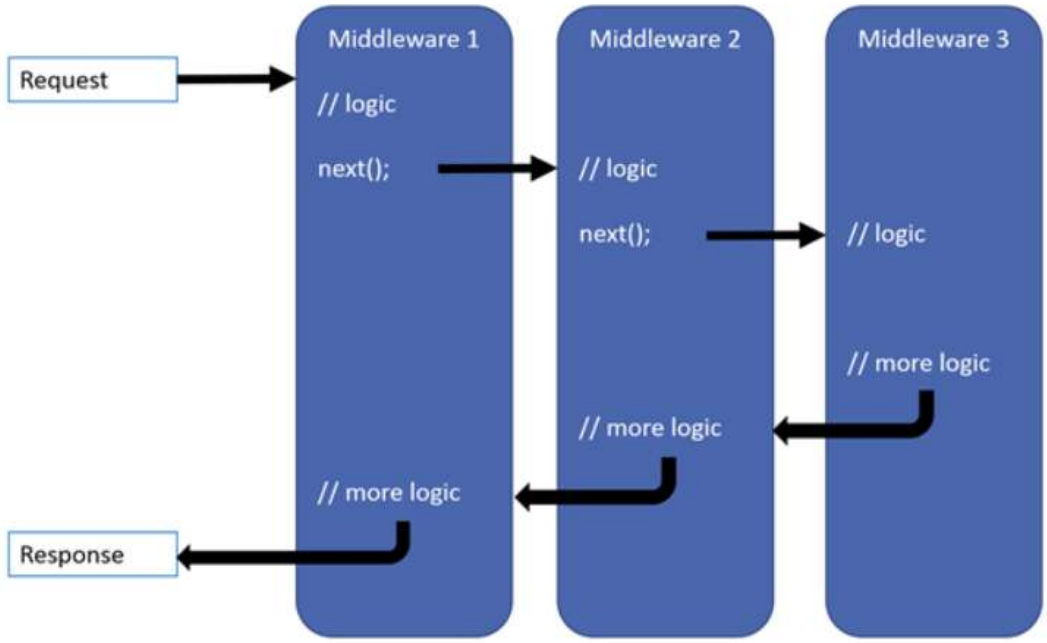

Рисунок 2.6 – Принцип роботи Middleware

2.2.1. Огляд рушія генерування веб-сторінок Razor

Razor - це мова розмітки для вбудовування серверного коду в веб-сторінки. Являє собою реалізацію Server Side Scripting для платформ ASP.NET та ASP.NET Core. Рушій, що виконує дану мову, також має назву Razor. Синтаксис Razor складається з розмітки Razor, C # і HTML. Файли, що містять Razor, мають розширення .cshtml. Рушій дозволяє використовувати мову C# для написання скриптів, які реалізують логіку генерації сторінок. Для відокремлення мови C# від мови HTML Razor використовую зарезервований символ @. Починаючи строку з заданого символу розробник починає писати на мові C#. Рушій та мова розмітки Razor дозволяють реалізувати такі конструкції як блоки, умовні конструкції та цикли. Приклад їх реалізації можна побачити на рисунках 2.7, 2.8, та 2.9.

```
@{
    var quote = "The future depends on what you do today. - Mahatma Gandhi";
}
```

Рисунок 2.7 – Приклад реалізації блоку на мові Razor

```
@if (value % 2 == 0)
{
    <p>The value was even.</p>
}
else if (value >= 1337)
{
    <p>The value is large.</p>
}
else
{
    <p>The value is odd and small.</p>
}
```

Рисунок 2.8 – Приклад реалізації умовних конструкцій на мові Razor

```
@for (var i = 0; i < people.Length; i++)
{
    var person = people[i];
    <text>Name: @person.Name</text>
}
```

Рисунок 2.9 – Приклад реалізації циклу на мові Razor

2.2.2. Огляд Entity Framework Core

Entity Framework Core (EF Core) являє собою об'єктно-орієнтовану, легковажну технологію від компанії Microsoft для доступу до даних. EF Core є ORM-інструментом.

ORM (Object-Relation-Mapping) – це технологія програмування, яка зв'язує бази даних (БД) з концепціями об'єктно-орієнтованих мов програмування, створюючи «віртуальну об'єктну базу даних».

Головною метою даного підходу є забезпечити роботу з даними в термінах класів, а не таблиць даних і навпаки, перетворити терміни і дані класів в дані, придатні для зберігання в системі управління базами даних (СУБД). Необхідно також забезпечити інтерфейс для CRUD-операцій над даними. Загалом, необхідно позбутися від необхідності писати SQL-код для взаємодії в СУБД [10].

CRUD (Create, Read, Update, Delete) – акронім, що позначає чотири базові функції, які використовуються при роботі зі сховищами даних:

- створення;
- читання;

- редагування;
- видалення.

Таким чином, EF Core дозволяє працювати з базами даних, але пропонує більш високий рівень абстракції, який дозволяє програмісту не зосереджувати увагу на самій БД та її таблицях, а працювати з даними незалежно від типу сховища. Якщо на фізичному рівні розробник оперує таблицями, індексами, первинними і зовнішніми ключами, але на концептуальному рівні, який нам пропонує EF, доводиться працювати з об'єктами.

Центральної концепцією Entity Framework є поняття сутності або entity. Сутність визначає набір даних, які пов'язані з певним об'єктом. Завдяки цьому дана технологія передбачає роботу не з таблицями, а з об'єктами та їх колекціями.

Будь-яка сутність, як і будь-який об'єкт з реального світу, має певні властивості. Наприклад, візьмемо сутність, яка описує людину. Тоді можна виділити такі властивості, як ім'я, зріст та вік. Вони відповідно будуть представлені такими типами даних, як string (рядок) чи int (ціле число). Звісно цими типами даних представлення полів не обмежується і вони можуть бути більш комплексними. У кожної сутності є мінімум одна унікальна властивість, яка дозволяє ідентифікувати її, така властивість називається ключем.

Між собою сутності можуть бути пов’язані асоціативним зв’язком. Цей зв’язок буває трьох типів, аналогічно зв’язкам таблиць у реляційних СУБД:

- один-до-одного;
- один-до-багатьох;
- багато-до-багатьох.

Однією з головних можливостей EF Core, як технології ORM є використання Language Intergrated Query (LINQ) для виконання запитів до БД. Entity Framework бере на себе транслювання LINQ-запиту у мову SQL. Таким чином розробник повністю абстрагується від бази даних, що дозволяє

створювати додатки, які зможуть працювати на різних СУБД без переписування основних модулів програми.

2.2.3. Огляд бібліотеки SignalR

Платформа ASP.NET Core має вбудовану бібліотеку, яка призначена для створення додатків, що працюють в режимі реального часу. Ця бібліотека має назву SignalR. Вона використовує двонаправлений зв'язок для обміну повідомленнями між клієнтом і сервером, завдяки чому сервер може відправляти в режимі реального часу всім клієнтам деякі дані. Дана бібліотека є реалізацію високого рівня абстракції технології виклику віддалених процедур.

Віддалений виклик процедур (Remote Procedure Call або RPC) – це клас технологій, що дозволяють комп'ютерним програмам викликати функції або процедури в іншому адресному просторі (як правило, на віддалених комп'ютерах). У більшості випадків SignalR забезпечує такий обмін повідомлень між сервером та клієнтом.

Бібліотека виконує спілкування між сервером та клієнтом одним з таких методів:

- за рахунок WebSocket;
- з використанням подій, що надсилаються сервером (Server-Sent Events);
- використовуючи Long Polling.

WebSocket – це протокол, який призначений для обміну інформацією між браузером та веб-сервером в режимі реального часу. Працює за рахунок протоколу TCP.

Для встановлення з’єднання надсилається HTTP-запит, який виконує роль «цифрового рукостискання» (рис 2.10). Клієнт також надсилає свій відкритий ключ в певному заголовку Sec-WebSocket-Key, який закодовано за технологією base64.

В разі встановлення з’єднання, сервером надсилається HTTP-відповідь, в якій надається підтвердження, що встановити з’єднання дозволено, а в

заголовку Sec-WebSocket-Accept міститься інформація про з'єднання, яка також кодується за технологією base64 (рис 2.11).

```
GET /ws HTTP/1.1
Host: pmx
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Version: 6
Sec-WebSocket-Origin: http://pmx
Sec-WebSocket-Extensions: deflate-stream
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
```

Рисунок 2.10 – Приклад HTTP-запиту на встановлення WebScoket з'єднання.

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
```

Рисунок 2.11 – Приклад HTTP-відповіді про дозвіл встановлення WebScoket з'єднання.

Server-Sent Events – це технологія відправки повідомлень сервером до веб-браузера у вигляді DOM-подій. Ця технологія входить до стандарту HTML5.

HTTP Long Polling – це методика опитування сервера клієнтом під час якого сервер зберігає запит відкритим, поки не буде сформовано відповідь. Після отримання відповіді клієнт знову відкриває з'єднання.

Бібліотека SignalR має можливість автоматично обрати тип можливого з'єднання. Пріоритетом є швидкодія, через це в першу чергу SignalR намагається використати WebScoket, потім SSE. Якщо жодний з цих типів з'єднання не доступний, то буде обрана методика HTTP Long Polling. Розробник також може самостійно обрати тип з'єднання.

2.3. Порівняння сучасних хмарних платформ

Хмарні обчислення (англ. Cloud Computing) – це модель забезпечення повсюдного та зручного доступу на вимогу через мережу до спільного пулу обчислювальних ресурсів, що підлягають налаштуванню (наприклад, до

комунікаційних мереж, серверів, засобів збереження даних, прикладних програм та сервісів), і які можуть бути оперативно надані та звільнені з мінімальними управлінськими затратами та зверненнями до провайдера [11].

Сьогодні хмарні обчислення набувають все більшої популярності. Багато великих сучасних компаній створили свої власні хмарні платформи і пропонують частину обчислювальної потужності своїх центрів обробки даних за певну плату. До таких компаній відносяться Google, IBM, Microsoft, Amazon та інші. Кожна з них має власне бачення того, як повинна виглядати хмарна платформа для свої клієнтів, через це кожна з них має як свою позитивні сторони так і негативні.

Розглянемо і проаналізуємо найпопулярніші хмарні платформи, які сьогодні розповсюджуються за схемою інфраструктура як послуга. Для цього порівняння було обрано такі три сервіси: Amazon Web Services (AWS), Google Cloud Platform та Microsoft Azure.

Компанія AWS була заснована в 2006 році. Amazon Web Services - це хмарна програма для побудови бізнес-рішень з використанням інтегрованих веб-служб. Ця компанія пропонує широкий спектр послуг за такими моделями: інфраструктура як послуга та платформа як послуга. До них відносяться Elastic Cloud Compute (EC2), Elastic Beanstalk, служба простого зберігання (S3) і служба реляційних баз даних (RDS).

AWS має такі позитивні риси:

- найбільша хмарна платформа;
- має багато регіонів по всьому світу;
- має безкоштовний пробний період;
- надійність, підтверджена великою кількістю клієнтів;

До недоліків можна віднести:

- складний в управлінні;
- ціна.

Google Cloud Platform – постачальник хмарних сервісів Google, який

було запущено в 2011 році. Платформа дозволяє користувачам створювати бізнес-рішення за допомогою модульних веб-сервісів Google. Він пропонує широкий спектр послуг, включаючи рішення інфраструктура як послуга і пдатформа як послуга.

Завдяки багатошаровій безпечній інфраструктурі Google Cloud користувачі можуть бути впевнені, що все, що вони будують, створюють або зберігають, буде захищено.

Google Cloud має безліч інструментів для забезпечення послідовної роботи та керування. До них відносяться Compute Engine, App Engine, контейнерний двигун, Cloud Storage і Big Query. Google також пропонує плавну міграцію на віртуальні машини.

Має такі переваги:

- ціна;
- активний розвиток;
- зручний у керуванні
- має безкоштовний пробний період.

Але можна виділити такі недоліки:

- менше функціональних можливостей у порівнянні з конкурентами;
- має менше регіонів в світі, ніж конкуренти.

Microsoft Azure - це набір хмарних сервісів, призначених для задоволення потреб середнього бізнесу. Він розроблений у 2010 році, щоб надати кожному бізнесу свободу для створення, керування та розгортання додатків у глобальній мережі, не примушуючи відмовлятися від своїх улюблених каркасів або інструментів.

Загалом для споживачів зараз доступно більше 100 різних продуктів Azure, починаючи від послуг машинного навчання AI до інструментів управління,, які виявляють, сортують та діагностують проблеми, що можуть бути присутніми у службах клієнтів. У поєднанні з інструментами та програмами сторонніх розробників Azure розроблено як універсальне місце для розробки.

Серед переваг даної платформи можна виділити:

- хороші параметри масштабованості;
- підтримка технологій Microsoft, серед яких акцентується увага на .NET та .NET Core;
- має найкраще покриття за кількістю регіонів;
- має безкоштовний пробний період.

До недоліків можна віднести:

- складний в управлінні;
- ціна.

Як видно з проведеного порівняння, сучасні хмарні платформи пропонують майже схожі функціональні можливості. У чомусь перемагає AWS, у чомусь Azure, а ,наприклад, Google Cloud перемагає в ціні. Для публікації хмарного сервісу для гри в шахи було обрано платформу Azure саме завдяки таким її критеріям, як легка масштабованість та краща підтримка стеку продуктів від Microsoft, серед яких і обраний фреймворк ASP.NET Core.

3. ХМАРНИЙ СЕРВІС ГРИ В ШАХИ З ІНТЕРАКТИВНИМ РЕЖИМОМ СПОСТЕРЕЖЕННЯ ЗА ГРОЮ

3.1. Загальна архітектура сервісу

Сьогодні існує багато архітектурних підходів для проектування клієнт-серверних додатків. Але найпопулярнішим залишається підхід під назвою багаторівнева архітектура.

Багаторівнева архітектура – це один з видів архітектури в програмній інженерії, в якій розділяються функції представлення, обробки і зберігання даних.

Найпопулярнішим різновидом багаторівневої архітектури є трирівнева архітектура, яка зображена на рис. 3.1. В основі її реалізації лежить відокремлення таких рівнів логіки:

1) рівень представлення, який відповідає за інтерфейс користувача та візуалізацію даних;
2) рівень бізнес-логіки, який відповідає за обробку даних;
3) рівень доступу до даних, який відповідає за збереження даних.

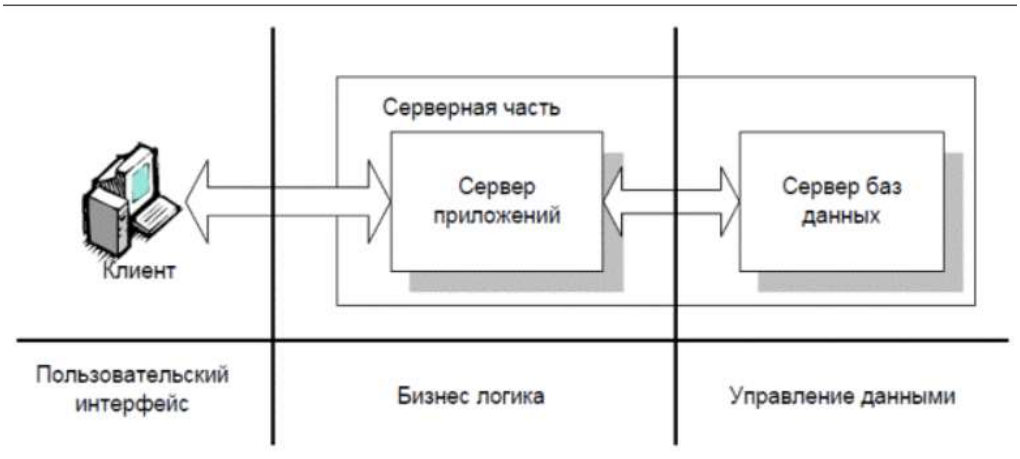

Рисунок 3.1 – Схема трирівневої архітектури інформаційної системи

Для забезпечення модульності розробленого додатку та полегшення процедури додавання нових функціональних можливостей було вирішено реалізувати дану архітектуру наступним чином.

Рівень роботи з даними було вирішено реалізувати через фреймворк Entity Framework Core та обгорнути логіку роботи з ним через патерн репозиторій (рис 3.2).

EF Core дозволяє забезпечити легкий перехід на іншу СУБД завдяки використанню підходу ORM.

Через те, що сервіс планується розмістити в хмарі, було вирішено обгорнути логіку роботи з даними в шаблон проектування під назвою «репозиторій». Більшість хмарних платформ (та Azure в тому числі) мають свої NoSQL сховища. Для платформи Azure – це Azure Tables та Azure Blobs. EF Core не вміє працювати з такими різновидами сховищ, таким чином у разі виникнення необхідності змінити сховище даних, патерн репозиторій дозволить не змінюючи основної логіки програми, реалізувавши логіку роботи з даними, виконати перехід до нового типу сховища даних.

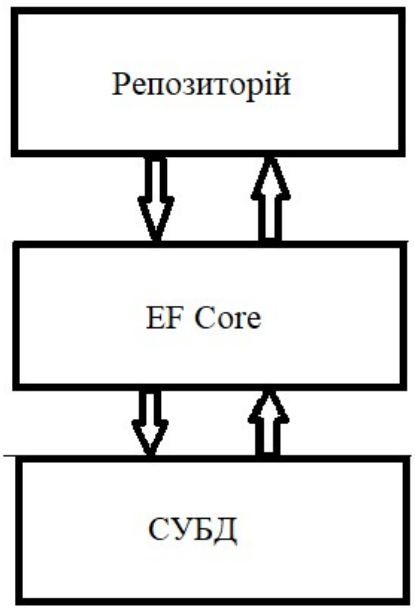

Рисунок 3.2 – Схема рівня роботи з даними

Для реалізації рівня бізнес логіки було вирішено обрати підхід під назвою Command Query Separation.

Command-query separation (CQS) або command-query responsibility segregation (CQRS) — це принцип імперативного програмування, винайдений Бертраном Мейером під час роботи над мовою програмування Eiffel[12].

Принцип вказує, що метод повинен бути командою, яка виконує дію над даними або запитом, який повертає дані. Таким чином запит не може змінювати дані, на відміну від команди. Такий підхід до роботи над даними пізніше перетворився у патерн проектування логіки роботи додатків.

Команда (Command) та запит (Query) реалізуються за допомогою такого типу даних, як структура чи клас. Для заданого сервісу було обрано структуру. Структура містить всі необхідні дані, які потрібно передати для виконання логіки команди або запиту. За обробку команд відповідає обробник команд (Command handler). За обробку запитів – обробник запитів (Query handler). Можуть бути створені декілька обробників для запитів або команд, кожний з яких, наприклад, може відповідати за роботу над певним типом даних (рис 3.3).

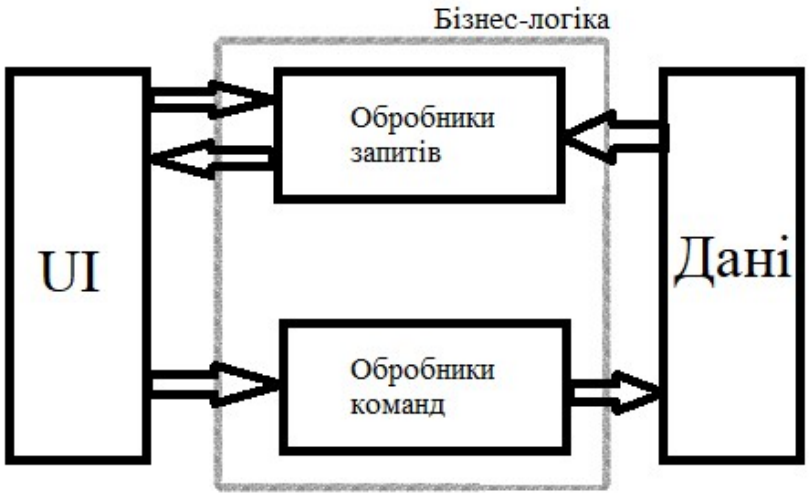

Рисунок 3.3 – Схема роботи бізне-логіки за патерном CQS

За реалізацію рівня представлення відповідає технологія Razor Pages. Це частина платформи ASP.NET Core, яка дозволяє відображати HTML сторінки, а також написати обробник до кожної HTML сторінки окремо. Дана технологія є альтернативою до технології ASP.NET Core MVC і була створена для полегшення написання веб-додатків, які використовують SSS підхід для відображення сторінок.

3.2. Структура бази даних

Проектування бази даних було виконано за методом Code First (спочатку код). Entity Framework може створити базу даних, відповідно до набору моделей сутностей, які представлені за допомогою класів на мові C#. Таким чином, достатньо розробити систему класів, які описують дані, якими буде оперувати сервіс, що розроблюється.

Відповідно до поставленого завдання у ТЗ, було створено такі основні класи:

- Article;
- Game;
- GameSubscription;
- Message;
- Chat;
- ChessPortalUser.

Клас Article відповідає сутності стаття та містить такі поля:

- Id – унікальний ідентифікатор статті;
- Url – посилання, яке відповідає даній статті;
- Title – заголовок статті;
- Content – текст статті;
- Image – посилання на головне зображення статті;
- TimePosted – дата й час, коли статтю було опубліковано;
- TimeUpdated – дата й час, коли останній час редагувалась стаття;

- PostedById – ідентифікатор користувача (адміністратора), який опублікував статтю;
- PostedBy – поле типу ChessPortalUser, яке вказує EF, що PostedById є зовнішнім ключем до таблиці користувачів сервісу.

Клас Game відповідає сутності гра (партія) та містить такі поля:

- Id - унікальний ідентифікатор гри;
- WhitePlayerId – ідентифікатор користувача, який грає білим кольором;
- WhitePlayer – поле типу ChessPortalUser, яке вказує EF, що WhitePlayerId є зовнішнім ключем до таблиці користувачів сервісу.
- BlackPlayerId – ідентифікатор користувача, який грає чорним кольором;
- BlackPlayer – поле типу ChessPortalUser, яке вказує EF, що BlackPlayerId є зовнішнім ключем до таблиці користувачів сервісу.
- GameStatus – поле типу int та відповідає за статус гри. Може відповідати таким значенням: очікування гравця, хід білих, хід чорних, гра закінчилась в нічию, гра закінчилась перемогою чорних/білих;
- IsPublic – поле типу bool та відповідає тому, чи гра є публічною або приватною;
- History – містить історію партію;
- CommonChatId – ідентифікатор публічного чату;
- CommonChat – поле типу Chat, яке вказує EF, що поле CommonChatId є зовнішнім ключем до таблиці чатів сервісу;
- PrivatChatId – ідентифікатор приватного чату;
- PrivatChat – поле типу Chat, яке вказує EF, що поле PrivateChatId є зовнішнім ключем до таблиці чатів сервісу;
- TimeCreated – зберігає дату та час створення гри;
- TimeUpdated – зберігає дату та час останнього оновлення гри;
- TimeEnded – зберігає дату та час кінця гри.

Клас GameSubscribtion відповідає сутності підписка та має такі поля:

- Id – унікальний ідентифікатор підписки;

- GameId – ідентифікатор гри, на яку підписались;
- Game – поле типу Game, яке вказує EF, що поле GameId є зовнішнім ключем до таблиці чатів сервісу;
- UserId – ідентифікатор користувача, який підписався на гру;
- User – поле типу ChessPortalUser, яке вказує EF, що поле UserId є зовнішнім ключем до таблиці користувачів сервісу.

Клас Message відповідає сутності повідомлення чату та містить такі поля:

- Id – унікальний ідентифікатор повідомлення;
- Text – містить текст повідомлення;
- FromId – ідентифікатор користувача, який надіслав повідомлення;
- From – поле типу ChessPortalUser, яке вказує EF, що поле FromId є зовнішнім ключем до таблиці користувачів сервісу;
- ChatId – ідентифікатор чату, до якого належить повідомлення;
- Chat – поле типу Chat, яке вказує EF, що поле ChatId є зовнішнім ключем до таблиці чатів сервісу;
- TimePosted – містить дату і час надіслання повідомлення.

Клас Chat відповідає сутності чат та містить такі поля:

- Id – унікальний ідентифікатор чату;
- Messages – поле типу ICollection<Message>, яке вказує EF, що таблиця чатів зв’язана з таблицею повідомлень за схемою one-to-many;

За сутність користувач відповідає клас ChessPortalUser. Так як для системи авторизації користувача ASP.NET Core має власну систему Identity, то для того, щоб користувач системи містив потрібні поля для авторизації достатньо успадкувати клас ChessPortalUser від класу IdentityUser.

Тоді клас ChessPortalUser буде містити такі основні поля:

- Id – унікальний ідентифікатор користувача;
- UserName – унікальне ім’я користувача;
- Email – електронна пошта користувача;
- PasswordHash – хеш паролю користувача;

- EmailConfirmed – поле типу bool, яке вказує чи підтвердив користувач свій email;
- AccessFailedCount – зберігає кількість невдалих спроб авторизації;
- LockoutEnd – містить час і дату розблокування користувача після невдалої спроби авторизації;
- IsBlocked – поле типу bool, яке показує чи заблокований користувач адміністратором;
- та інші поля, використання яких буде додано у майбутньому.

Структуру бази даних описує клас ChessPortalContext, який успадковано від класу IdentityDbContext.

Клас ChessPortalContext містить такі поля:

- Users – поле типу DbSet<ChessPortalUser>, що відповідає за таблицю користувачів;
- Games – поле типу DbSet<Game>, що відповідає за таблицю партій;
- GameSubscribtions – поле типу DbSet<GameSubscribtion>, що відповідає за таблицю підписок;
- Messages – поле типу DbSet<Message>, що відповідає за таблицю повідомлень;
- Chats – поле типу DbSet<Chat>, що відповідає за таблицю чатів;
- Articles – поле типу DbSet<Article>, що відповідає за таблицю статей;
- та інші, які створюються системою Identity.

Для того, щоб створити або оновити після зміни моделі базу даних, потрібно спочатку створити міграцію. Функція міграції в EF Core забезпечує спосіб поступового оновлення схеми бази даних для синхронізації з моделлю даних програми, зберігаючи існуючі дані в БД. Для цього достатньо з папки проекту виконати команду dotnet ef migrations add <назва міграції>. Після створення міграції, потрібно виконати dotnet ef database update і база даних буде оновлена. Схема створеної бази даних зображено на кресленні в додатку А.

3.3. Розподілений менеджер блокувань

Основною проблемою написання веб-додатків, які планується масштабувати, є синхронізація потоків. Серверна частина таких додатків у більшості випадків опрацьовує запити клієнтів асинхронно. Завдяки цьому значно пришвидшується робота додатку при великому навантаженні.

Але виникають виключні ситуації, коли користувачу необхідно виконати операцію, коректність роботи якої залежить від того, який потік перший її виконає, а доступ с початку і до кінця операції повинен бути заборонений для інших потоків. При масштабуванні додатків, коли копія програми запускається на різних машинах (в одному чи на декількох кластерах), стандарті засоби синхронізації не працюють. Для цього використовують розподілений менеджер блокувань.

Розподілений менеджер блокувань (Distributed Lock Manager або DLM) - це пакет програмного забезпечення, який дозволяє комп'ютерам в кластері синхронізувати свої доступи до спільно використовуваних ресурсі.

Для проектування власного менеджеру блокувань було вирішено обрати Redis для зберігання інформації про ресурс, до якого потрібно заборонити доступ.

Redis – це розподілене сховище пар ключ-значення, які зберігаються в оперативній пам'яті, з можливістю забезпечувати довговічність зберігання за бажанням користувача. Дане програмне забезпечення має відкритий вихідний код.

Розробники даного сервісу пропонують власний алгоритм розподіленого блокування під назвою RedLock, який було використано в дипломному проекті для реалізації розподіленого менеджеру блокувань [14]. Розглянемо спочатку випадок, коли ми маємо єдиний екземпляр запущеного Redis. Для того, щоб поставити блокування, клієнт надсилає команду, проілюстровану на рис. 3.5. Команда встановить ключ лише в тому

```
SET resource_name my_random_value NX PX 30000
```

Рисунок 3.5 – Команда встановлення блокування для ресурсу в Redis

випадку, якщо вона ще не існує (опція NX), з терміном дії 30000 мілісекунд (опція PX). Ключ встановлюється у значення "myrandomvalue". Це значення має бути унікальним для всіх клієнтів і всіх запитів блокування.

В основному випадкове значення використовується для того, щоб вивільнити блокування безпечним способом, за допомогою скрипта, який повідомляє Redis: видалити ключ, тільки якщо він існує, а значення, що зберігається в ключі, точно є таким, який я очікую. Це досягається завдяки скрипту на мові Lua, який зображено на рис. 3.6.

```
if redis.call("get",KEYS[1]) == ARGV[1] then
    return redis.call("del",KEYS[1])
else
    return 0
end
```

Рисунок 3.6 – Скрипт на мові Lua для вивільнення блокування

Це важливо, щоб уникнути видалення блокування, створеного іншим клієнтом. Наприклад, клієнт може придбати замок, заблокувати його в певній операції довше, ніж час дії блокування (час, коли закінчиться термін дії ключа), а потім видалити блокування, яке вже було придбано іншим клієнтом. Використання операції DEL без перевірки не є безпечним, оскільки клієнт може видалити блокування іншого клієнта. За допомогою цього сценарію замість кожного блокування «підписується» випадковий рядок, тому блокування буде видалено, тільки якщо воно залишається тим, що було встановлено клієнтом, який намагається його видалити.

Тепер розглянемо випадок, коли ми маємо розподілену версію БД Redis у хмарі. Тоді, спираючись на реалізацію блокування для одного екземпляру, клієнт має виконати наступні кроки, щоб виконати блокування для всіх:

1. Взяти поточний час в міллісекундах;
2. Спробувати послідовно отримати блокування у всіх N екзеплярах, використовуючи одне і те ж ім'я ключа і випадкове значення. Під час кроку 2, при встановленні блокування в кожному випадку, клієнт використовує тайм-аут, який є невеликим порівняно з загальним часом автоматичного вивільнення блокування. Наприклад, якщо час автоматичного вивільнення становить 10 секунд, час очікування може бути в діапазоні ~ 5-50 мілісекунд. Це запобігає блокуванню клієнта протягом тривалого часу, намагаючись розмовляти з вузлом Redis, який не працює: якщо екземпляр недоступний, клієнт повинен спробувати поговорити з наступним екземпляром;
3. Обчислити, скільки часу минуло для того, щоб отримати блокування, вирахувавши з поточного часу мітку часу, отриману на кроці 1. Якщо і тільки якщо клієнт зміг придбати замок у більшості випадків , і загальний час, що минув для придбання блокування, менше, ніж час дії блокування, блокування вважається виконаним;
4. Якщо блокування виконано, то час його дії вважається початковим часом дії мінус час, що минув, який був обчислений на кроці 3.
5. Якщо клієнт з якоїсь причини не зміг виконати блокування (або він не зміг заблокувати таку кількість екземплярів: [N / 2 + 1], або час дії є негативним), він спробує розблокувати всі екземпляри.

Коли клієнт не може отримати блокування, він повинен спробувати знову після випадкової затримки, яка використовується з метою десинхронізувати декілька клієнтів, які намагаються придбати блокування для одного і того ж ресурсу одночасно (це може призвести до розблокування ситуації, коли ніхто не може отримати).

Вивільнення блокування є простим і передбачає просто відпускання блокування у всіх випадках, незалежно від того, вважає клієнт, що він зможе успішно заблокувати даний екземпляр чи ні.

3.4. Проектування та розробка сервісу

Для шахового сервісу було створено діаграму прецедентів згідно технічного завдання. На основі цієї діаграми було реалізовано програмну частину сервісу.

Діаграма прецедентів – в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі. Також, перекладається як діаграма варіантів використання.

Розглянемо діаграму прецедентів, яку було створено для шахового сервісу. На рисунку 3.7 зображено акторів системи. Користувач сервісу в загалом може поділятись на авторизованого та неавторизованого. Авторизовані користувачі у свою чергу поділяються на гравця та адміністратора. На основі цього для системи було створено такі ролі користувачів як “user” (гравець) та “admin” (адміністратор).

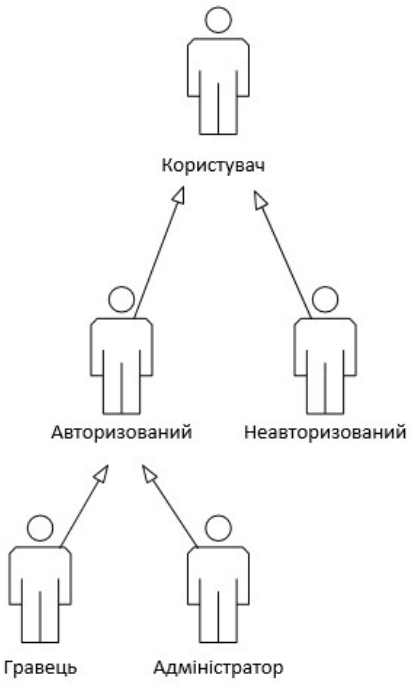

Рисунок 3.7 – Діаграма акторів шахового сервісу

Кожен з цих типів користувачів (неавторизований, гравець та адміністратор) мають доступ до певних функціональних можливостей сервісу.

Для неавторизованого користувача доступні основні функції, які зображено на рисунку 3.8. Користувач може переглянути статті на сайті, переглянути партії, які грають в даний момент часу та авторизуватись в системі. Під час авторизації користувач може увійти в систему чи зареєструватись. Після реєстрації виконується вхід в систему.

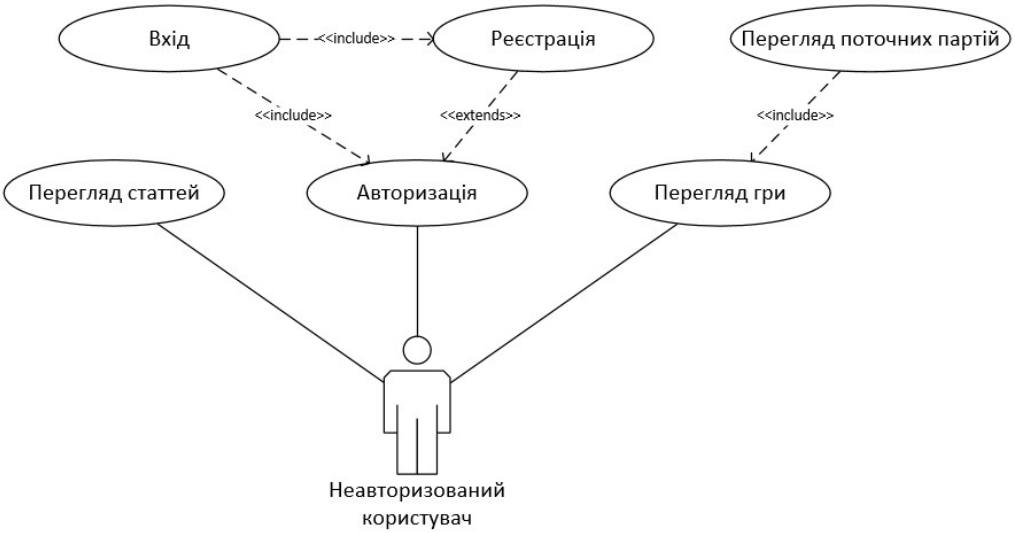

Рисунок 3.8 – Діаграма доступних функціональних можливостей сервісу для неавторизованого користувача

Розглянемо можливості авторизованого користувача, а саме – гравця. Авторизований користувач має такі ж можливості, як і неавторизований, але авторизація замінюється виходом з системи. З рисунку 3.9 видно, що гравець має можливість керувати партіями, а саме створювати нові партії та грати в існуючі. Наявна можливість управління акаунтом з редагуванням персональних даних. Під час перегляду гри авторизований користувач має додаткові можливості, як писати в чаті та підписатися на гру.

Рисунок 3.9 – Діаграма доступних функціональних можливостей для гравця

Адміністратор шахового сервісу має як додаткові, так і такі ж функціональні можливості, що і гравець. На рисунку 3.10 зображено додаткові можливості для адміністратора шахового сервісу. Даний тип користувача може редагувати, видаляти та створювати статті для сервісу, а також блокувати гравців за порушення.

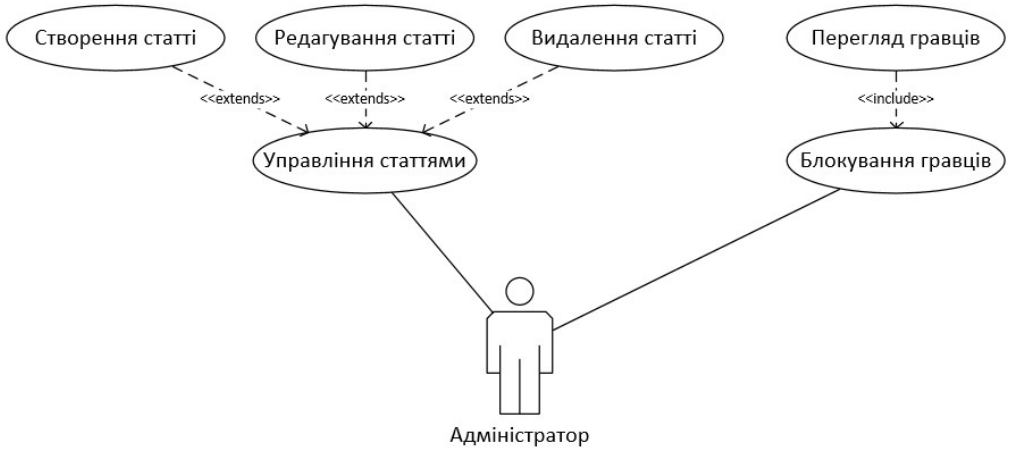

Рисунок 3.10 – Діаграма додаткових функціональних можливостей для адміністратора

Завдяки тому, що для реалізації логіки додатку було обрано патерн CQS, реалізувати всі описані вище функціональні властивості сервісу буде відносно легко.

Розглянемо на прикладі «перегляду гри». Для того, щоб користувачу відобразити партію, потрібно завантажити її з БД. Ця подія є запитом (Query). Приклад її реалізації, а також обробника цього запиту можна побачити на рис. 3.11 та 3.12 відповідно.

```
public struct GetFullGameInfoQuery : IRequest<Game>
{
    public readonly string Id;

    public GetFullGameInfoQuery(string id)
    {
        Id = id;
    }

}
```

Рисунок 3.11 – Запит на завантаження гри з БД

```
public Task<Game> Handle(GetFullGameInfoQuery request, CancellationToken cancellationToken)
    => _repository.AsQueryable()
                .Include(g => g.WhitePlayer)
                .Include(g => g.BlackPlayer)
                .FirstOrDefaultAsync(g => g.Id == request.Id);
```

Рисунок 3.12 – Обробник запиту на завантаження гри з БД

Нехай користувач авторизований, тоді він може написати в чат. Збереження повідомлення в чат є командою (command), так як це змінює стан системи. Приклад реалізації команди зображений на рис. 3.13, 3.14, та 3.15.

```
public struct CreateCommand<TModel> : IRequest<CommandResult<TModel>> {
    public readonly TModel Model;

    public CreateCommand(TModel model) {
        Model = model;
    }
}
```

Рисунок 3.13 – Команда створення об’єкту

```
public async Task<CommandResult<Message>> Handle(CreateCommand<Message> request, CancellationToken cancellationToken) {
    request.Model.TimePosted = DateTime.UtcNow;

    var result = await _messageRepository.AddAsync(request.Model);
    return new CommandResult<Message>(CommandResultStatus.Executed, result);
}
```

Рисунок 3.14 – Обробник команди створення повідомлення

```
model.FromId = Context.User.GetUserId();
var message = AutoMapper.Mapper.Map<MessageViewModel, Message>(model);

var result = await _mediator.Send(new CreateCommand<Message>(message));
```

Рисунок 3.15 – Приклад виклику команди

Додаток було вирішено розділити на 3 основних та 3 додаткових модулі (або пакети), які зображено на рис 3.16. Основні модулі представляють рівень роботи з даними (ChessPortal.Data), рівень логіки додатку (ChessPortal.Logic) та рівень представлення (ChessPortal.Web). Додаткові модулі містять спільні константи та ресурси (ChessPortal.Common), спільні сервіси (ChessPortal.Services) та додаткові теги для сторінок Razor (ChessPortal.TagHelpers).

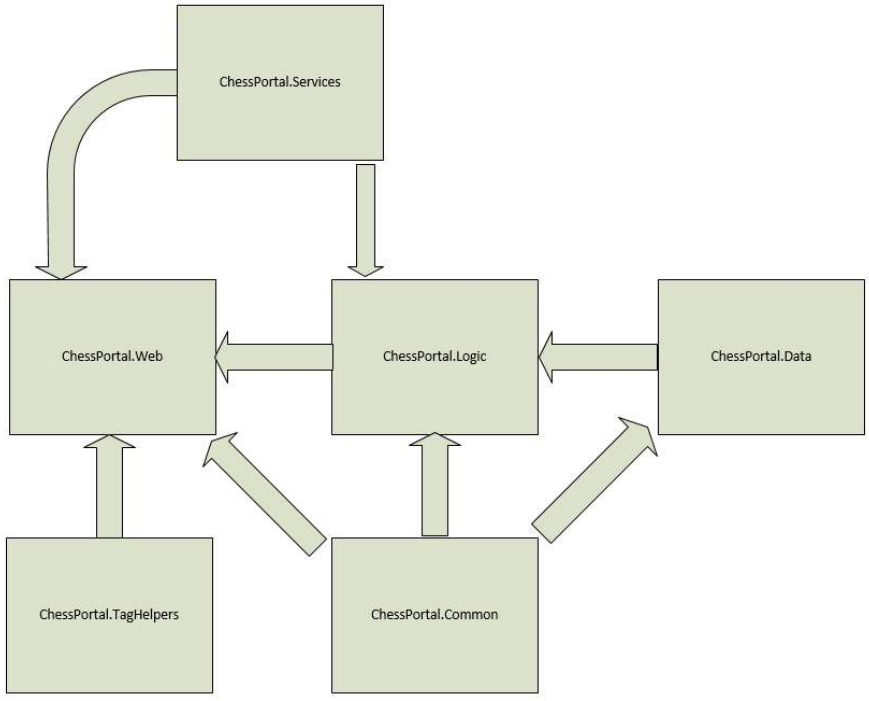

Рисунок 3.16 – Взаємодія модулів додатку

4. ІНТЕРФЕЙС КОРИСТУВАЧА ТА ТЕСТУВАННЯ СЕРВІСУ

4.1. Інтерфейси користувачів

4.1.1. Інтерфейс неавторизованого користувача

При вході в систему всі користувачі потрапляють на головну сторінку. З неї можна перейти у розділ новин або поточних партій. Також користувач може авторизуватись чи зареєструватись у сервісі.

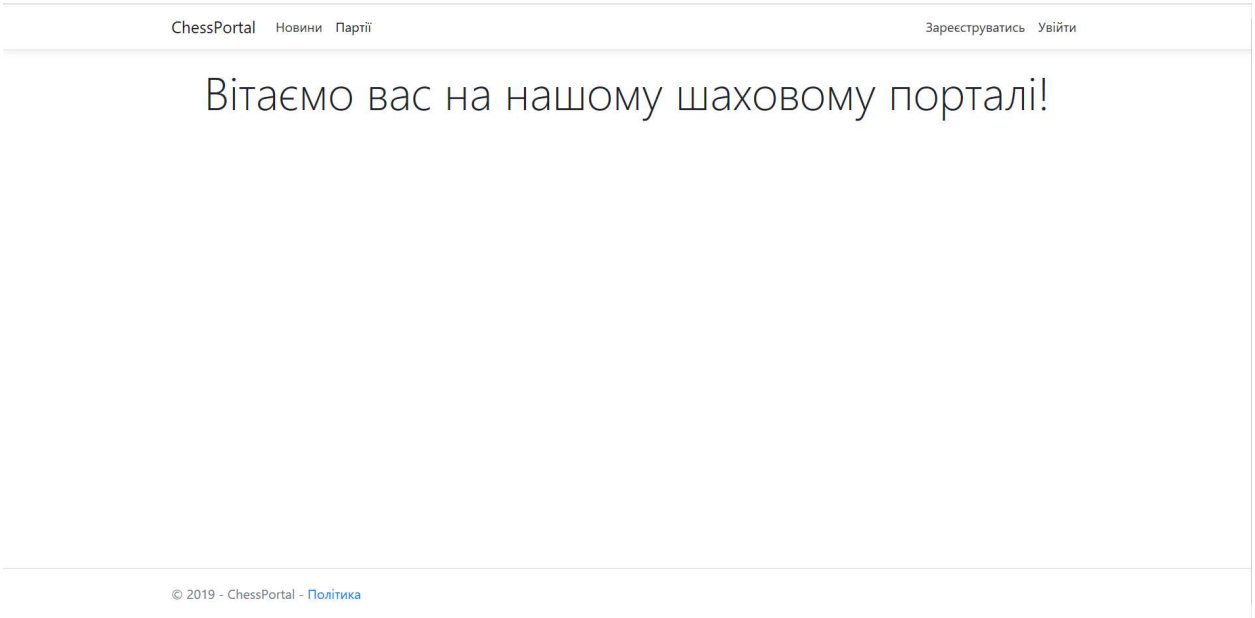

Рисунок 4.1 – Головна сторінка порталу

На сторінці партій користувач може обрати партію, за якою він хоче стежити (рис 4.2). Функціональні можливості поточної партії для неавторизованого глядача будуть обмеженими, так як для підписки та можливості писати у чаті необхідно авторизуватись (рис 4.3).

На сторінці авторизації користувач може увійти в систему через логін та пароль, або скористатись альтернативним методом авторизації через провайдер таких соціальних мереж, як Facebook чи Google (рис 4.4). У випадку, якщо користувача не існує, сервіс запропонує зареєструватись, прив’язавши до аккаунта обраний провайдер.

На сторінці реєстрації користувач може зареєструватись у системі (рис 4.5).

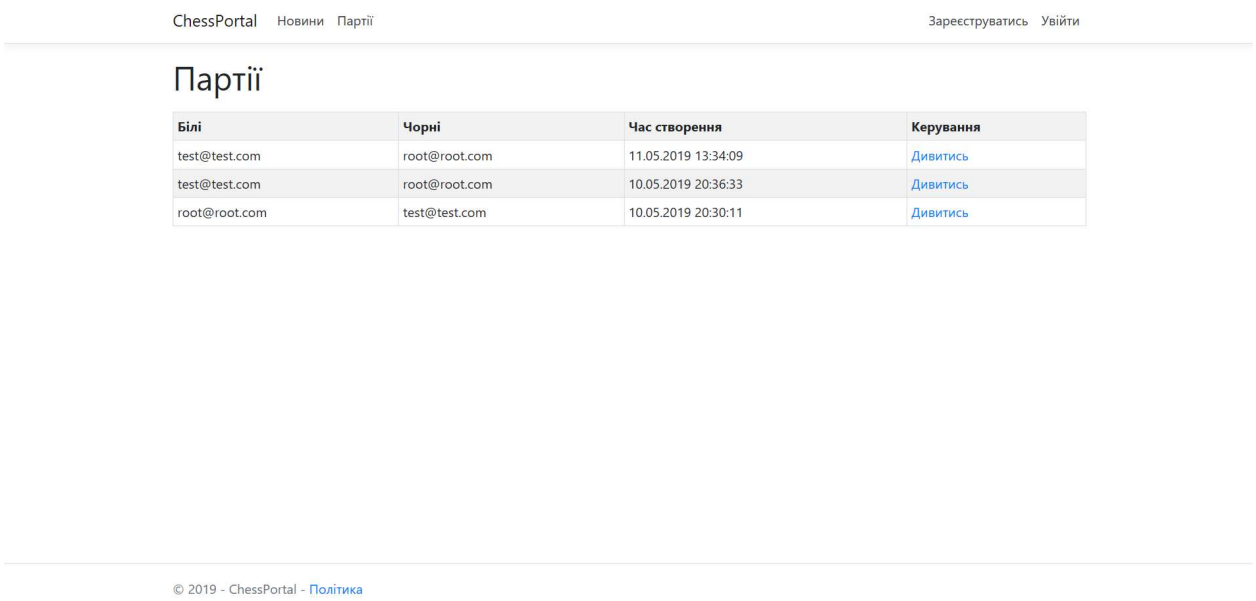

Рисунок 4.2 – Сторінка активних публічних партій

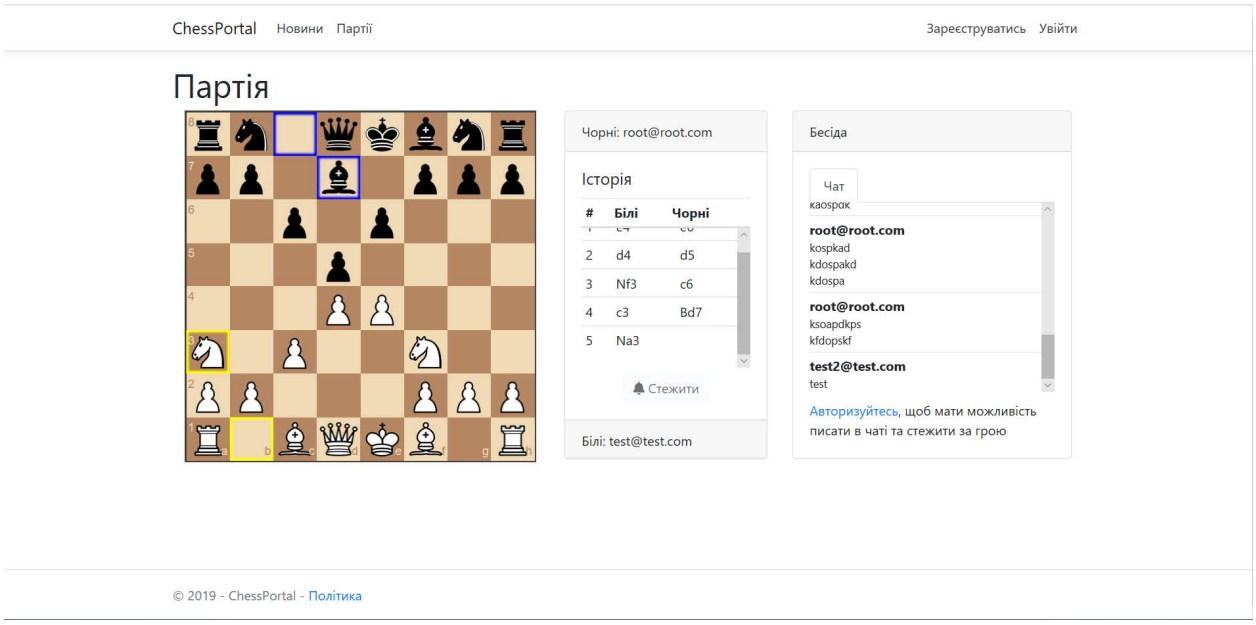

Рисунок 4.3 – Сторінка гри для неавторизованого глядача

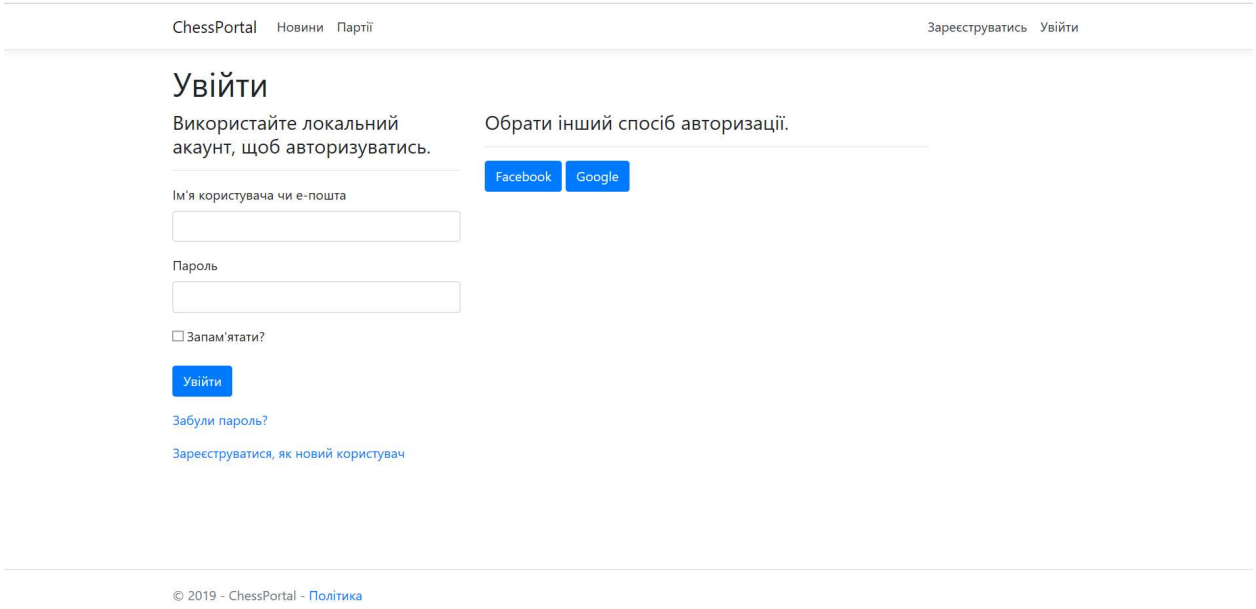

Рисунок 4.4 – Сторінка авторизації

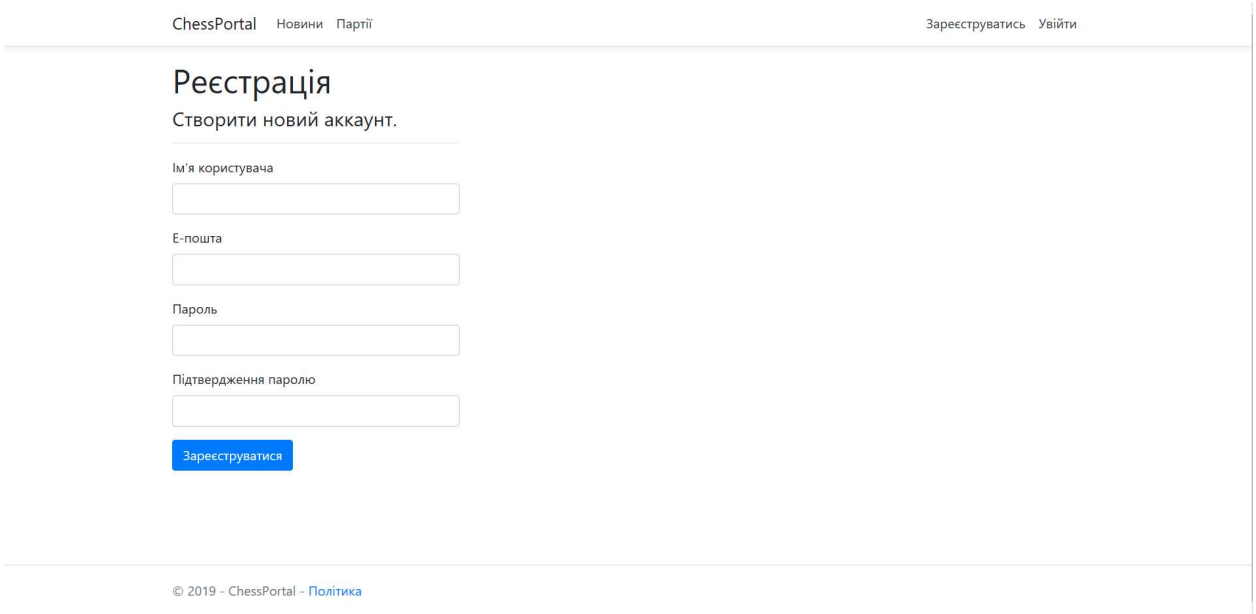

Рисунок 4.5 – Сторінка реєстрації

4.1.2. Інтерфейс гравця

Авторизувавшись, гравець отримує доступ до управління його партіями (рис 4.6), а також до редагування персональних даних (рис 4.7).

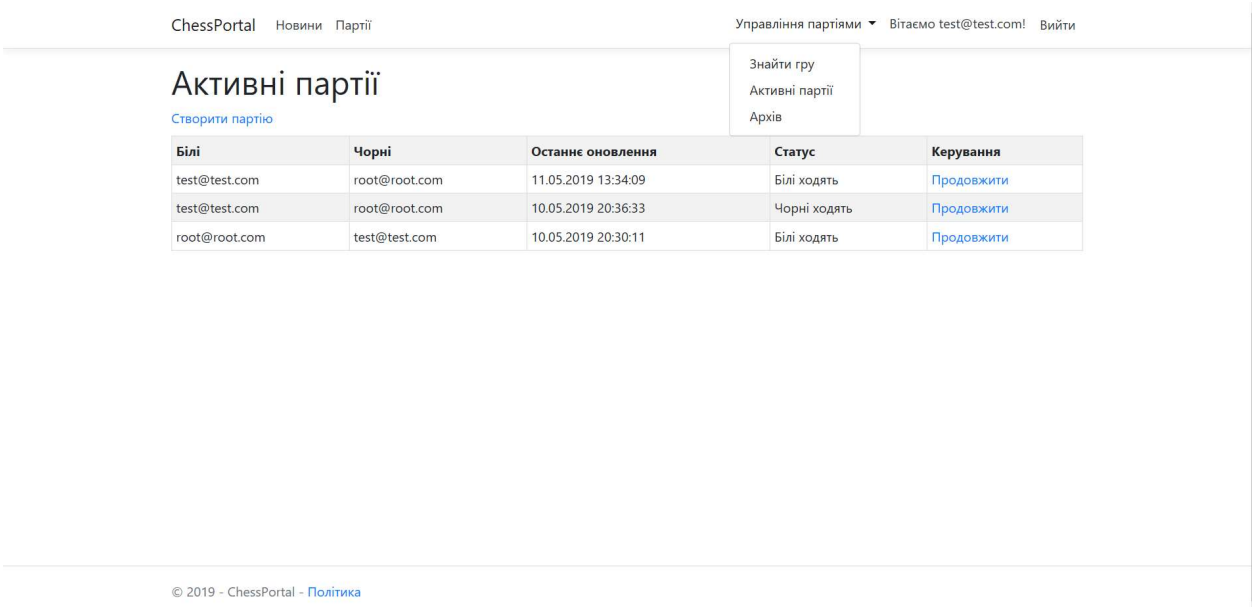

Рисунок 4.6 – Сторінка активних партій

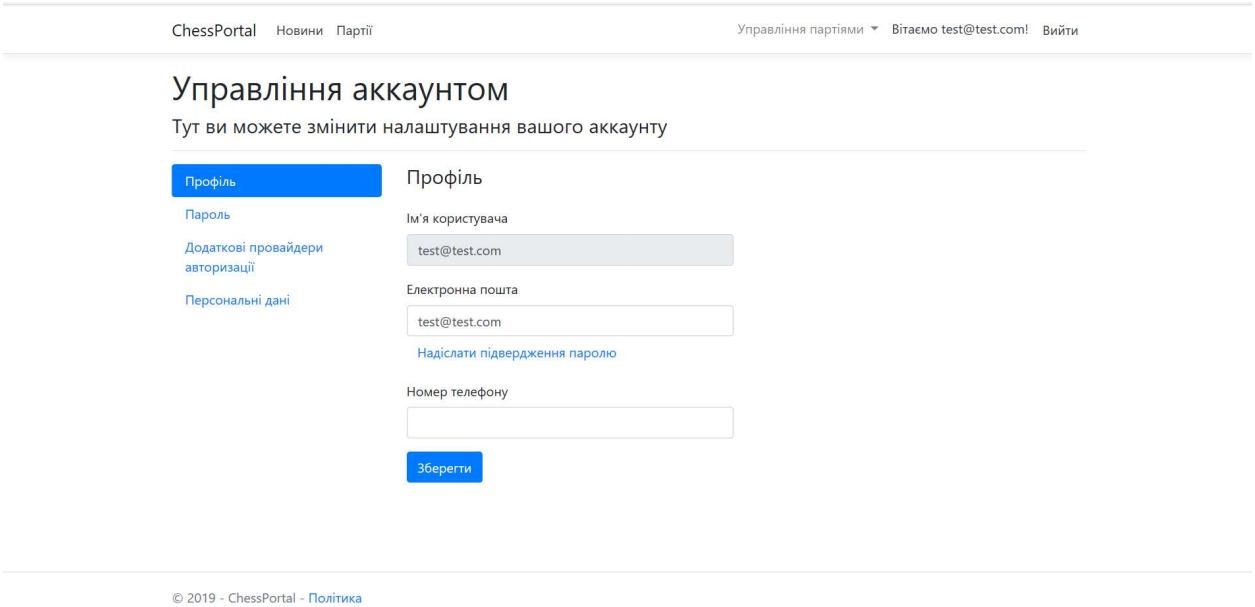

Рисунок 4.6 – Сторінка управління аккаунтом

Під час партії гравець може писати повідомлення в приватний та публічні чати, здатися або запропонувати нічию (рис 4.7).

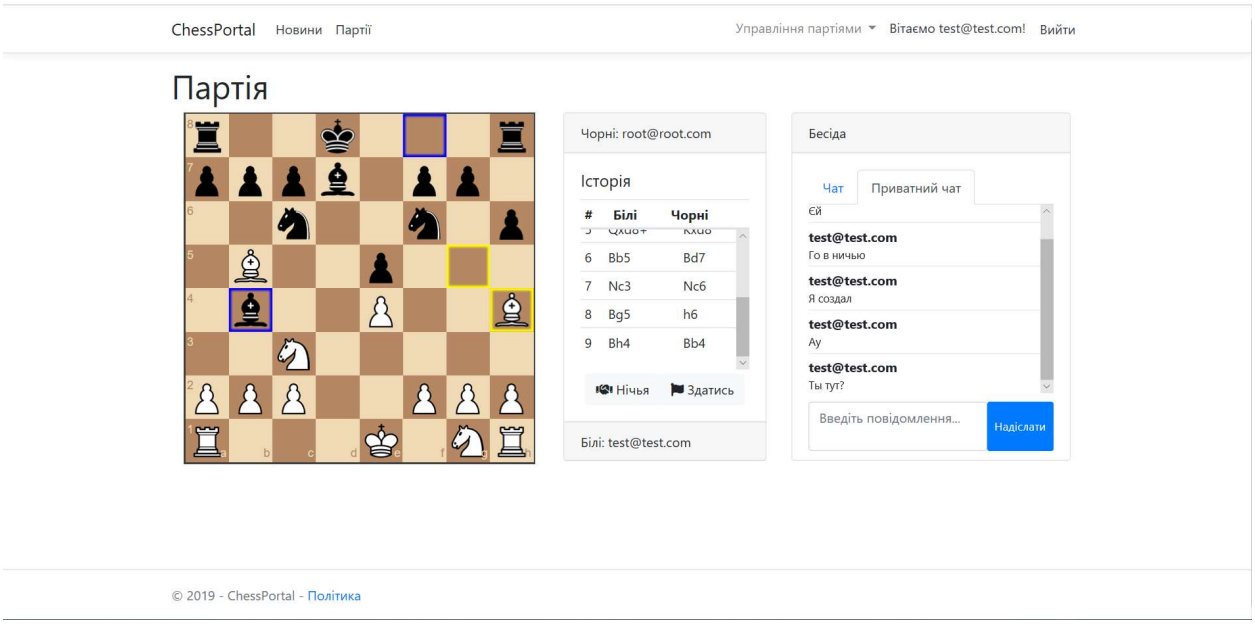

Рисунок 4.7 – Інтерфейс гравця під час гри

На сторінці архівних партій (які вже завершені) користувач може завантажити на комп'ютер або переглянути онлайн (рис 4.8).

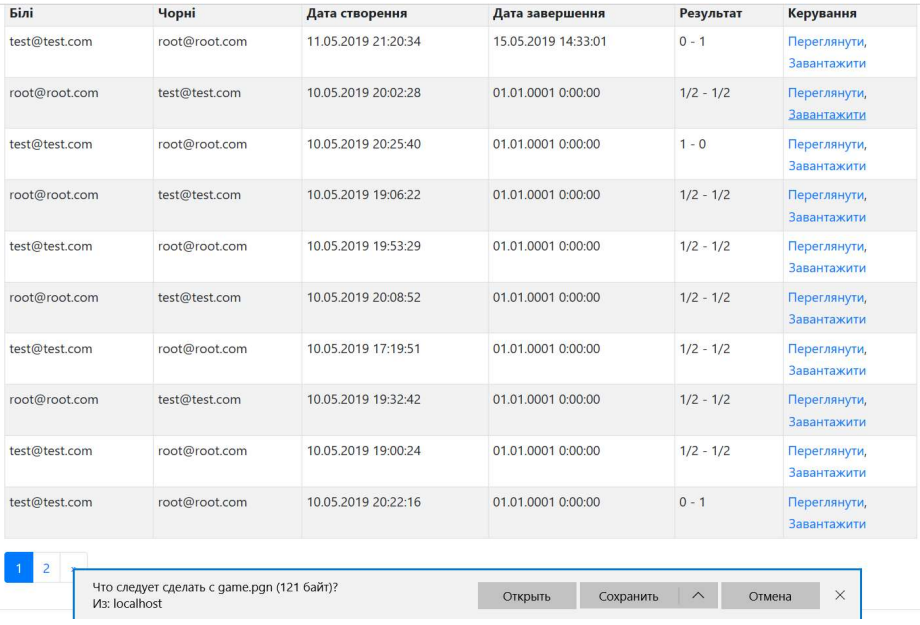

Рисунок 4.8 – Архів партій

Якщо зареєстрований користувач є глядачем, то він зможе підписатись на партію (або відписатись), що буде сповіщено відповідним повідомленням (рис. 4.9).

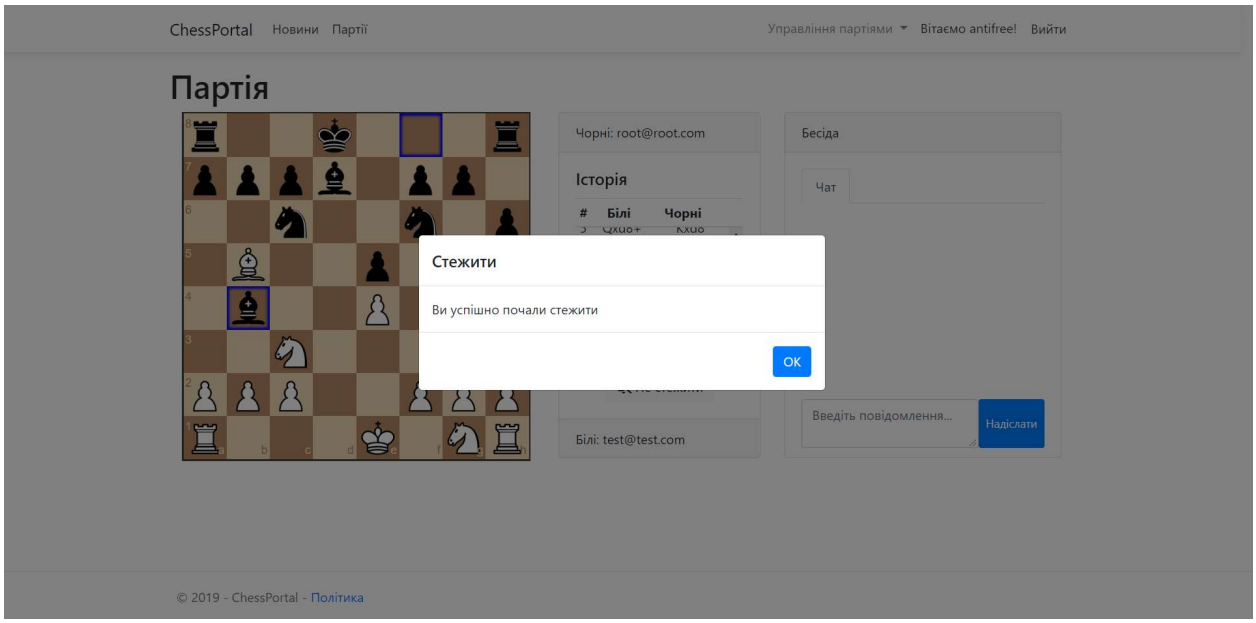

Рисунок 4.9 – Сповіщення про початок стеження за грою

4.1.3. Інтерфейс адміністратора

Адміністратор має додаткові функціональні можливості у вигляді менеджменту статей та блокуванню/розблокуванню користувачів.

Сторінку керування статтями зображено на рисунку 4.10. Адміністратор

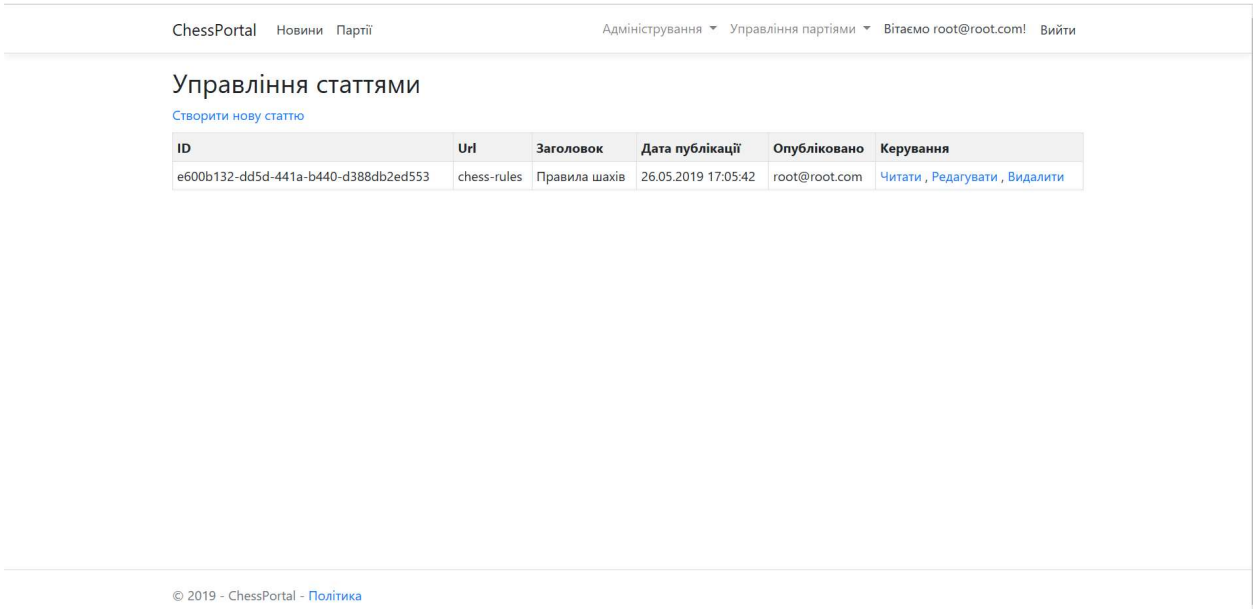

Рисунок 4.10 – Сторінка менеджменту статей

має можливість статтю (рис 4.11), редагувати (рис 4.12), видалити, а також створити нову статтю.

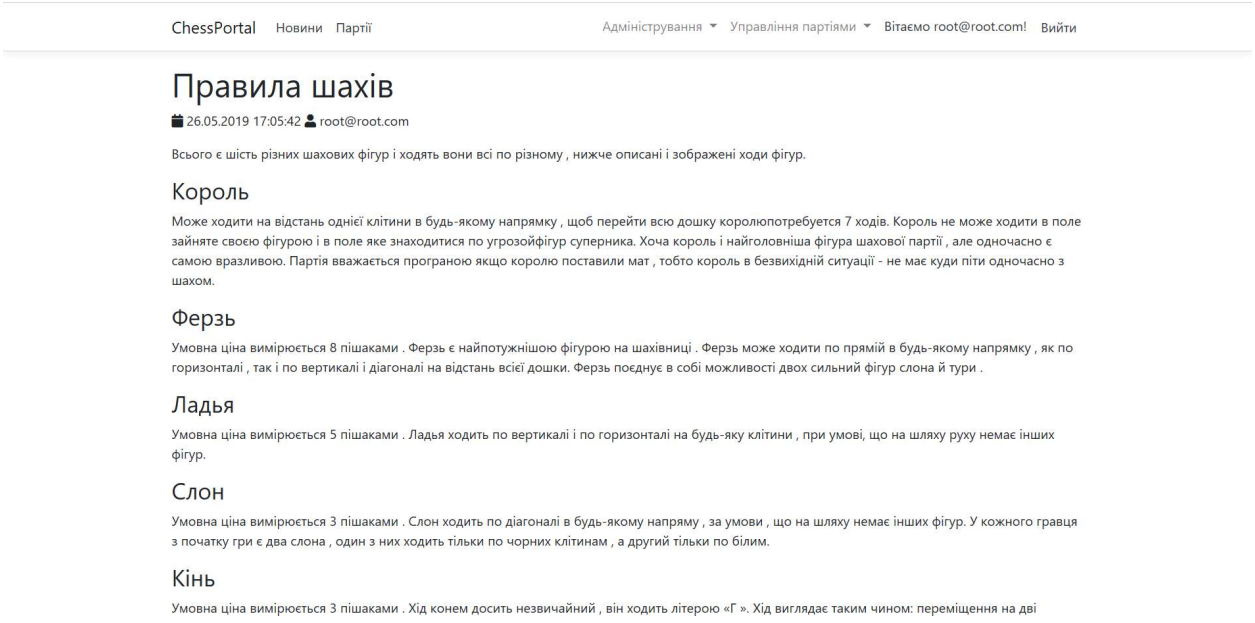

Рисунок 4.11 – Відображення статті на порталі

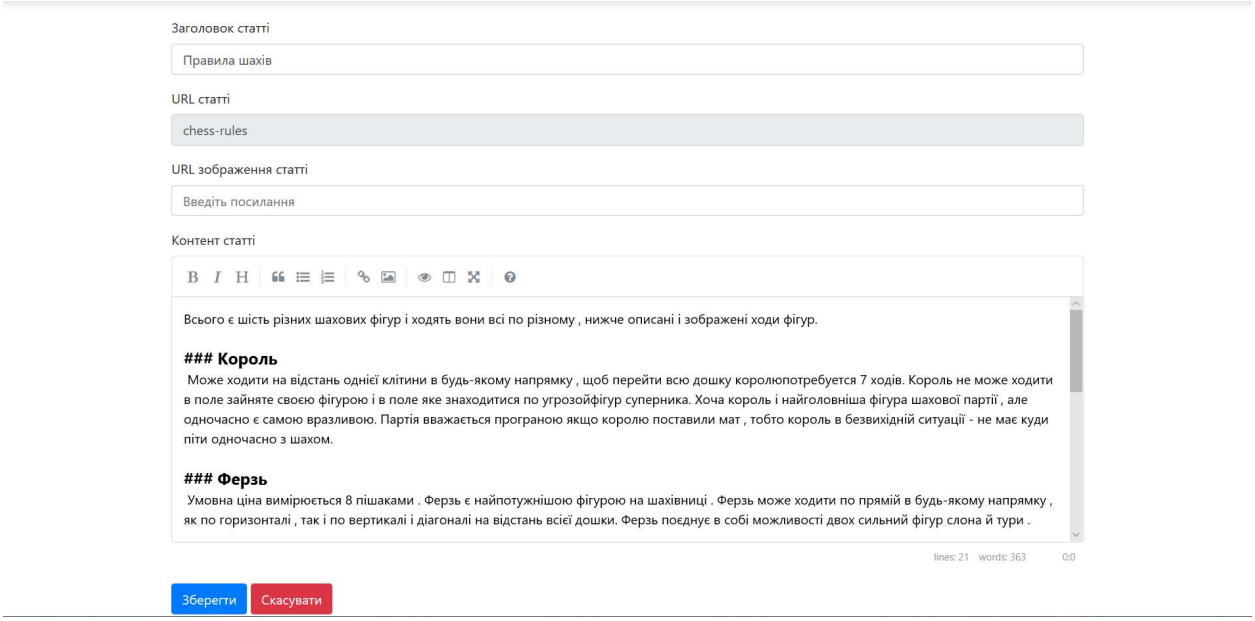

Рисунок 4.12 – Редагування статті на порталі

Адміністратор сервісу може заблокувати гравця за порушення, якщо такі будуть виявлені, на сторінці з користувачами, яка зображена на рисунку 4.13.

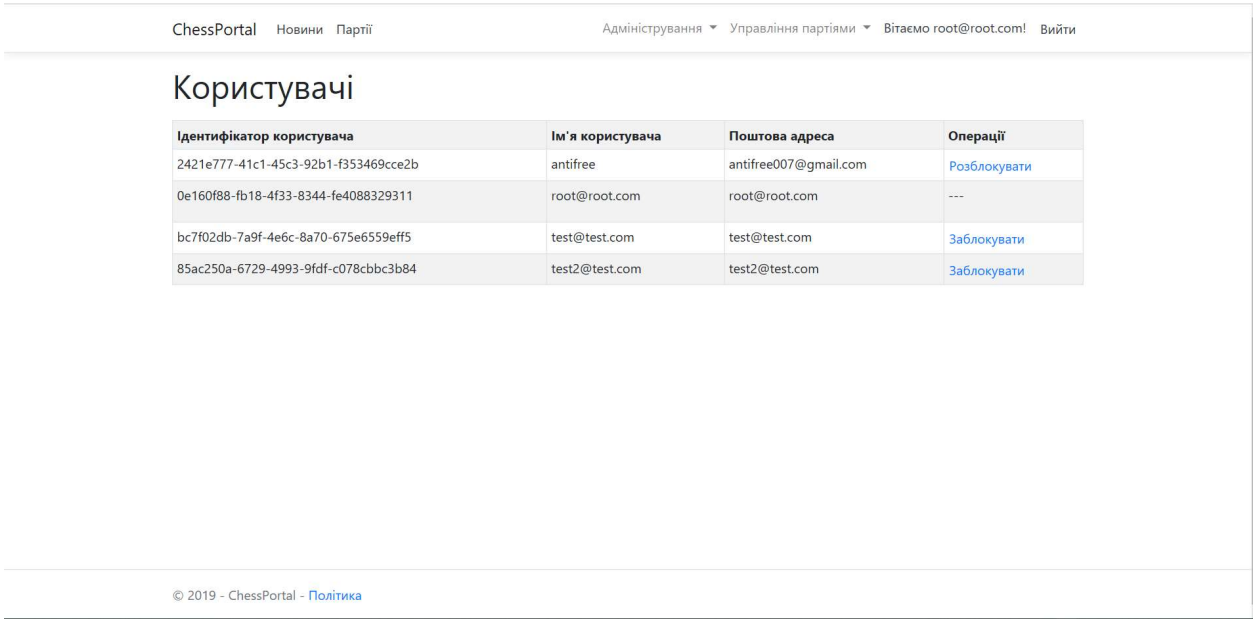

Рисунок 4.13 – Сторінка управління користувачами

Інтерфейс розробленого додатку є адаптивним, таким чином сервісом зручно керувати як з комп'ютеру, так і з мобільного пристрою (рис 4.14).

Рисунок 4.14 – Інтерфейс додатку для мобільних пристроїв

4.2. Тестування сервісу

Сьогодні існує багато різновидів тестування програмного забезпечення. Виділяють такі основні категорії тестів:

- за метою;
- за ступенем автоматизації;
- за рівнем;
- за формальністю;

Тестування за метою поділяється на функціональне та нефункціональне. Функціональне тестування відповідає за перевірку наявності і коректності роботи усіх функціональних властивостей програмного продукту. Нефункціональне тестування перевіряє коректність роботи не функціональних можливостей системи, наприклад, тестування стабільності роботи.

Тестування за рівнем автоматизації поділяється на ручне та автоматизоване. Ручне тестування здійснюють розробники та тестувальники. Автоматизоване тестування виконується автоматично спеціальним програмним модулем, який використовується для виявлення помилок в коді, які призводять до некоректної роботи тестів.

Тестування за рівнем поділяється на модульне (unit), інтеграційне та системне тестування. Модульне тестування відповідає за перевірку роботи окремого модуля. Інтеграційне тестування здійснює перевірку спільної роботи декількох модулів системи. Системне тестування перевіряє всю систему в цілому.

Тестування за формальністю поділяється на тестування за тестами, дослідницьке тестування та вільне тестування.

Розроблений шаховий сервіс було вирішено тестувати за метою, а саме обрати функціональне тестування. Відповідність виконання усіх функціональних властивостей системи, що були описані в ТЗ, можна побачити у попередньому розділі (див. 4.1).

Для автоматизації тестування важливих функціональних можливостей

системи було вирішено написати інтеграційні тести. Для тестування коректності роботи системи достатньо протестувати правильність сумісної роботи рівня логіки та рівня роботи з даними.

Під час тестування створюється база даних, що зберігається у оперативній пам’яті (in-memory) та заповнюється тестовими даними. Такий підхід дозволяє не навантажувати тестовий сервер (комп’ютер) повноцінною громіздкою БД, але при цьому відбувається повноцінна емуляція роботи з даними.

Для реалізації автоматизованих тестів був обраний фреймворк xUnit та бібліотека FluentAssertions.

xUnit.net – це безкоштовний інструмент для тестування модулів з відкритим вихідним кодом. Написана винахідником NUnit v2, xUnit.net є новою технологією для модульного та інтеграційного тестування C #, F #, VB.NET та інших .NET мов.

Тести написані для кожного з обробників команд та запитів, які створюються через DI перед початком тестування. Робота кожного запиту та команди перевіряється відповідними даними, які повинні бути при правильному результаті роботи. У випадку, якщо дані не співпадають, виникає повідомлення про помилку.

ВИСНОВКИ

Метою дипломного проекту була реалізація хмарного сервісу гри в шахи з інтерактивним режимом спостерігання за грою.

Через відсутність аналогів такого сервісу українського виробництва, а також велику популярність шахів, було вирішено розробити даний сервіс. Перед розробкою було розглянуто і проаналізовано існуючі рішення та враховано їх функціональні можливості.

Для розробки веб-додатку було використано трьохрівневу клієнт-серверну архітектуру. Для реалізації рівня логіки було обрано шаблон проектування CQRS, а в якості СУБД було використано MS SQL Server. За реалізацію онлайн гри відповідає бібліотека SignalR. З метою виконання тестування були написані інтеграційні тести, які покривають основні функціональні властивості системи.

Сервіс розміщено на хмарній платформі в тестовому режимі. Для локального запуску можна використати такі сервери, як IIS Express чи Kestrel.

Для покращення функціональності системи, а саме для кращого аналізу зіграних партій, в подальшому, на клієнтській стороні планується інтеграція шахового двигуна Stockfish у WebAssembly реалізації. Також планується додати глядачам можливість прокручувати всі ходи партії. Це допоможе краще зрозуміти гру, у випадку, якщо глядач не бачив її початок. З метою полегшити навчання шахам планується ввести режим розв’язування шахових задач.

Даний сервіс спрямований на підвищення популярності шахів в нашій країні, а також на об’єднаня шахістів в одне велике співтовариство. Розроблені та майбутні функціональні можливості сервісу будуть корисними як професійним гравцям, так і початківцям. Завдяки розташуванню на хмарній платформі, сервіс буде легко масштабуватись з ростом кількості його користувачів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Гери Д., Хорстманн К. JavaServer Faces – М.: Вильямс, 2008. – 576 с.
2. Б. Кришнамурти, Дж. Рексфорд. Web-протоколы. Теория и практика. — М.: ЗАО «Издательство БИНОМ», 2002 г. — 592 с.
3. Мэтт Зандстра. PHP: объекты, шаблоны и методики программирования 3-е издание. — М.: «Вильямс», 2010. — 560 с.
4. Доусон М. Программируем на Python. — СПб.: Питер, 2012. — 432 с.
5. У. Чан, П. Биссекс, Д. Форсье. Django. Разработка веб-приложений на Python = Python Web Development with Django // пер. с англ. А. Киселёв. — СПб.: Символ-Плюс, 2009. — 456 с.
6. Nathan Rozentals. Mastering TypeScript - Build enterprise-ready, industrial strength web applications using TypeScript and leading JavaScript Frameworks. — Packt Publishing, 2015. — 694 с.
7. Рихтер Дж. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.:Питер, 2013. — 896 с.
8. Марк Дж. Прайс. C# 7 и .NET Core. Кросс-платформенная разработка для профессионалов. 3-е изд. — СПб.: Питер, 2018. — 640 с.
9. Dino Esposito. Programming ASP.NET Core. — Microsoft, 2018. — 1131с.
10. Ноубл, Дж., Андерсон, Т., Брэйтуэйт, Г., Казарио, М., Третола, Р. Flex 4. Рецепты программирования. — БХВ-Петербург, 2011. — 720 с.
11. Peter Mell, Tim Grance. The NIST Definition of Cloud Computing // NIST: NIST — 2011, 3с.
12. Meyer, Bertrand. Object-oriented Software Construction. — Prentice Hall, 1988. — 1254 с.
13. Julia Lerman, Rowan Miller. Programming Entity Framework: Code First. — Highway North, Sebastopol: O'Reilly Media Inc., 2012 — 178с.
14. RedLock Algorithm – [Електронний ресурс] – 2015 – Режим доступу: https://redis.io/topics/distlock – Дата доступу: квітень 2019.

15. Пол Киммел. UML. Основы визуального анализа и проектирования. / Пол Киммел. – М.: НТ-Пресс, 2008. – 272 с.

16. Пол Киммел. UML. Универсальный язык программирования / Пол Киммел. – М.: НТ-Пресс, 2008. – 272 с.

17. Новиков Ф. А. Моделирование на UML. Теория, практика, видеокурс / Ф. А. Новиков, Д. Ю. Иванов. – М.: Наука и техника, 2010. – 640 с.